

Entwicklung von Werkzeugen zur Datenbereinigung in relationalen Datenbanken

Bachelorarbeit
im Studiengang Informatik

Philipp Wolski
Mtr.-Nr. 2518220

Fachgebiet Datenbanken und Informationssysteme
Institut für Praktische Informatik
Fakultät für Elektrotechnik und Informatik
Gottfried Wilhelm Leibniz Universität Hannover

Prüfer: Prof. Dr. Udo Lipeck
Zweitprüfer: Dr. H. H. Brüggemann
Betreuer: Prof. Dr. Udo Lipeck

1. April 2009

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Zielsetzung und Anforderungen	2
1.3	Aufbau der Arbeit	2
2	Grundlagen	5
2.1	Der Begriff Datenbereinigung	5
2.1.1	Relevante Datengegenstände	5
2.1.2	Hierarchien innerhalb der Datengegenstände	8
2.2	Ablauf einer Bereinigung	9
2.3	Verfahren zur Fehlererkennung	11
2.3.1	Attribut-Ebene	12
2.3.2	Tupel-Ebene	13
2.3.3	Objekt-Ebene	14
2.4	Verfahren zur Fehlerkorrektur	14
2.4.1	Korrektur mit Referenzwert	14
2.4.2	Merging	15
2.5	Bibliographischer Datenbestand des DBIS	16
3	Ausgewählte Verfahren	19
3.1	Überblick	19
3.2	Q-Grams	20
3.3	Field-Similarity	22
3.4	Vergleich von Q-Gram zu Field-Similarity	22
3.5	PC-Filter	23
4	Konzeption	31
4.1	Überblick	31
4.2	Architektur	31

4.2.1	Anwendung des MVC-Musters	33
4.2.2	Blockstruktur	34
4.2.3	Datenspeicher	35
4.3	Ablaufbeschreibung	36
4.4	Werkzeuge	39
4.4.1	Tupelwerkzeuge	40
4.4.2	Attributwerkzeuge	43
4.5	Mergingverfahren	45
4.6	Referenzwertverfahren	46
4.7	Ähnlichkeitsverfahren	47
4.7.1	Boolscher Vergleich	48
4.7.2	Nummerischer Vergleich	48
4.7.3	Datums-Vergleich	49
4.7.4	Modifizierte Field-Similarity	49
4.7.5	ISBN Vergleich	52
4.8	Prozessunterbrechung	53
5	Benutzerhandbuch	55
5.1	Installation und Ausführung	55
5.1.1	Anforderungen	55
5.1.2	Vorbereitungen zur Installation	56
5.1.3	Ausführung	56
5.2	Hauptoberfläche: Konfiguration	56
5.2.1	Datenbankzugang	58
5.2.2	Tabellenkonfiguration	59
5.2.3	Attributkonfiguration	60
5.2.4	Prozessliste	61
5.2.5	Werkzeugeinstellungen	62
5.3	Hauptoberfläche: Prozess	65
5.4	Interaktion	66
6	Testbericht	69
6.1	Vorbereitung	69
6.2	Laufzeittest	71

7 Implementierung	73
7.1 Umgebung	73
7.2 Entwurf und Aufbau	73
7.2.1 Paketübersicht	74
7.2.2 Klassenübersicht	75
7.2.3 Aufbau von Werkzeuge	81
7.3 Implementation vom Prozesszustand	82
7.4 Implementation von Werkzeugen	83
7.4.1 Klasse AttributReplacer	83
7.4.2 Klasse TupelDuplicate	84
7.5 Implementation von Verfahren	85
7.5.1 Wertunabhängige Verfahren	85
7.5.2 PC-Filter	85
7.5.3 Mergingverfahren	87
7.5.4 Referenzwertverfahren	88
7.5.5 Modifizierte Field-Similarity	88
8 Ausblick	91
8.1 Erweiterung von Tupeln auf Objekte	91
8.2 Erweiterung der Werkzeuge	91
8.3 Sicherung der Datenqualität	92
A API	93
A.1 Paket Configuration	93
A.2 Paket View	96
A.3 Paket Tools	99
Abbildungsverzeichnis	101
Beispielverzeichnis	103
Listenverzeichnis	105
Tabellenverzeichnis	107
Literaturverzeichnis	109

Kapitel 1

Einleitung

1.1 Motivation

Zur Archivierung einer Vielzahl von Daten setzt man heute bevorzugt Datenbanksysteme ein. Solche Systeme sind in der Regel sehr leistungsstark im Management der Datensätze und erlauben das Abspeichern unterschiedlichster Datentypen. Weiterhin erlauben es relationale Datenbanken verschiedene Datentabellen in Beziehung zu setzen, um Datensätze mit weiterführenden Informationen zu verknüpfen. Damit lassen sich sehr schnell sehr große und komplexe Datenspeicher realisieren. Bringt man in ein solches System ein fehlerbehaftetes Element ein, ist schnell einleuchtend, dass sich dessen Fehler auf den Datenbestand auswirken. Ein solches Element ist unter anderem der Mensch.

Die Eingabe von Daten in ein Datenbanksystem geschieht in der Regel über vom Menschen geschaffene Kanäle. Entweder werden Daten per Hand eingegeben oder es werden Importmechanismen genutzt, deren Logik sich auf vom Menschen vorgegebene Regeln stützt. Solange bei der Konversion der Quelldaten über den Kanal in das Datenbanksystem die von der Datenstruktur des Datenbanksystems implizierten bzw. die von dem Verwender intendierten Darstellung beibehalten wird, stellt dies kein Problem dar. Durch Unwissenheit oder Unachtsamkeit des Eingebenden, sowie durch unbehandelte Ausnahmen bei der Umwandlung von Quelldaten können allerdings Daten in das Datenbanksystem gelangen, deren Aufbau solch einer Darstellung widerspricht. Resultiert dies in einer Verletzung einer explizit oder implizit überwachten Eingaberegeln, z.B. Wertebereiche oder Datentypen, verweigert vermutlich das Datenbanksystem die Eingabe. Ist die Verletzung aber nur semantischer Struktur und wird nicht durch eingebaute Eingaberegeln überwacht, können fehlerhafte Einträge zurückbleiben. Dies kann geschehen durch vertauschte Buchstaben in Buchtiteln, andere Schreibweise für gleiche Wörter oder durch Fehler in der automatischen Neuformatierung von Quelldaten. Das Resultat solcher Fehler sind gleichbedeutende *Attributwerte* in unterschiedlichen Datensätzen mit abweichender Schreibweise oder sogar *duplizierte Datensätze*.

Fehler in einer komplexen Datenbank zu beheben kann per Hand sehr aufwändig sein. Da solche Systeme meist darauf ausgelegt sind, eine große Anzahl an Einträgen zu verarbeiten, läge es nahe, dem Datenbanksystem diese Aufgabe zu überlassen. Dazu müssen Methoden entwickelt werden, mit denen das System in der Lage ist, in annehmbarer Zeit selbständig datenbezogene Fehler erkennen und beheben zu können. Solche Methoden findet man in der Literatur unter den Begriffen *Data Cleaning* und *Data Cleansing* und

ordnet solche in die Thematik *Sicherung der Datenqualität* [BS06] ein. Der Vorgang einer solchen Fehlerbereinigung umfasst wie schon angedeutet die *Fehlererkennung* und anschließend die *Fehlerkorrektur*.

Die Verfahren zur Fehlererkennung müssen insbesondere bei der Suche nach duplizierten Daten in der Lage sein, sowohl die inhaltliche Gleichheit von Attributwerten zu beurteilen, als auch die von ganzen Datensätzen zu erkennen und dies auch in Beziehung gesetzten Tabellen. In der Regel erfordert dies eine Spezialisierung auf den gewählten Datenbestand, insbesondere auf die definierten *Eingabeformate* oder die inhaltliche Bedeutungen von Werten die durch *Domämentypen* definiert sind.

Nach der Fehlererkennung wird es mitunter Fälle geben, in denen entschieden werden muss, welche Daten als *korrekt* gelten. Eine solche Entscheidung zu treffen und anschließend die Daten zu korrigieren ist Bestandteil der Fehlerkorrektur.

Im Falle duplizierter Datensätze ist es nötig zu entscheiden, welche Informationen aus den Datensätzen den resultierenden Datensatz bilden. Das dafür nötige Verfahren wird im Folgenden auch *Merging* genannt.

1.2 Zielsetzung und Anforderungen

In dieser Arbeit sollen Werkzeuge entwickelt werden, die den Benutzer oder Datenadministrator bei der Bereinigung von relationalen Datenbanken unterstützen, sowie Maßnahmen vorgestellt werden, die zur zukünftigen Sicherung der Datenqualität beitragen.

Dazu sollen dem Benutzer Werkzeuge an die Hand gegeben werden, die ausgesuchte Verfahren der Literatur umsetzen und vom Benutzer auf seine Datenbank eingestellt werden können oder, mit etwas Programmieraufwand, auf eigene Datentypen oder Eingabeformate erweitert werden können. Dabei soll vom Benutzer, abgesehen von der Konfiguration auf den Datenbestand, möglichst wenig Interaktion erforderlich sein. Ebenso sind für den Benutzer geeignete Voreinstellungen für Konfigurationsmöglichkeiten zu wählen. Als besonderes Problem gilt es hierbei, die Erkennung von Fehlern in zueinander in Beziehung gesetzten Tabellen zu ermöglichen und diese automatisch zu korrigieren oder Korrekturvorschläge zur Wahl zu stellen, bei denen die Datensatzinformationen bereits bestmöglich für den Benutzer aufgearbeitet wurden.

Insbesondere wird dieses Werkzeugset zur Bereinigung der vom Fachgebiet Datenbanken und Informationssysteme geführten bibliographischen Datenbank entwickelt. Es sind daher schwerpunktmäßig Funktionen für das Bereinigen dieser bibliographischen Daten sowie notwendiger Datentypen und Eingabeformate zu entwickeln.

Die Werkzeuge sind speziell für ein Oracle Datenbank Management System ab Version 10 zu entwickeln. Das erlaubt die Ausnutzung von leistungsfähigen datenbankbasierten Programmen.

1.3 Aufbau der Arbeit

Diese Arbeit setzt sich zusammen aus der Einführung in die Grundlagen der Datenbereinigung, der Vorstellung von Verfahren sowie der Konzeption und der Implementierung der zu entwickelnden Werkzeuge.

Kapitel 2 beschäftigt sich mit den Grundlagen der Datenbereinigung, führt die in der Arbeit verwendeten Begriffe ein und gibt eine Übersicht über die speziellen Gegebenheiten der Testdatenbank.

Das Kapitel 3 stellt die in dieser Arbeit aus der Literatur übernommenen Verfahren vor und erläutert deren Funktionsweise.

Kapitel 4 beschäftigt sich mit der Konzeption der Werkzeuge: dazu wird auf die Entwicklungsumgebung eingegangen, die Erweiterungen an den zuvor vorgestellten Verfahren werden beschrieben und die Prinzipien der Werkzeuge dargestellt.

Kapitel 5 enthält das Handbuch zu den entwickelten Werkzeugen.

Kapitel 6 zeigt einen Beispielprozess in der Praxis und die Probleme, die dabei auftreten können.

Kapitel 7 beschreibt die Implementierung der in Kapitel 4 entwickelten Werkzeugsuite.

Das Kapitel 8 gibt einen Ausblick auf weitere Möglichkeiten der Datenbereinigung.

Im Anhang ist die API-Dokumentation zu der im Kapitel 6 beschriebenen Implementierung zu finden.

Kapitel 2

Grundlagen

Als ersten Schritt wird sich dieses Kapitel mit den Grundlagen der Datenbereinigung befassen. Dazu werden zunächst die zu betrachtenden Datengegenstände vorgestellt, die bei einer Bereinigung eine Rolle spielen. Anschließend werden Umfang und Ablauf einer Bereinigung vorgestellt und Beispielverfahren genannt, die bei einer Bereinigung zum Einsatz kommen können. Abschließend wird die bibliographische Datenbank des Fachgebiets Datenbanken und Informationssysteme vorgestellt.

2.1 Der Begriff Datenbereinigung

Der Begriff *Datenbereinigung* kann sehr allgemein verwendet werden. Je nachdem was als Daten und was als Datenverunreinigung angesehen wird, ändert sich der Blickwinkel hinsichtlich des Begriffs *Datenbereinigung*. Zum Beispiel kann das Entfernen überflüssiger Leerzeichen in einem Textdokument eine Datenbereinigung sein, aber auch das Auffinden und Korrigieren von fehlerhaften Einträgen in Datenbanken. Die gleichzeitige Erwähnung der Begriffe „*Auffinden*“ und „*Korrigieren*“ zeigt, dass sich eine Bereinigung sowohl aus **Fehlererkennung** als auch aus **Fehlerkorrektur** zusammensetzt. Dies ist bereits ein Hinweis darauf, dass neben der Definition, was bereinigt werden soll, auch der Umfang bzw. die Tiefe der Operationen zu definieren ist, nach deren Ausführung der Datenbestand als bereinigt angesehen werden soll. Es stellen sich folglich die Fragen:

- Was sind die Daten?
- Was sind Verunreinigungen in den Daten?
- Was soll mit den Verunreinigungen geschehen?

Statt von Verunreinigungen kann man hier auch von Fehlern sprechen. Im Weiteren sei für diese Arbeit der Begriff *Bereinigung* mit *Datenbereinigung* in relevanten Abschnitten gleichbedeutend.

2.1.1 Relevante Datengegenstände

Die Aufgabenstellung der Arbeit schränkt den Raum, in dem eine Antwort auf die zuvor genannten Fragen gefunden werden kann, auf relationale Datenbanken ein. Auf den ersten

Blick scheint demnach die Frage nach den Daten mit Datensätzen in einer solchen Datenbank beantwortet. Betrachtet man aber nun die Informationen in einem Datensatz, stellt man vor allem bei komplexen Datensatzzusammenhängen fest, dass der Begriff „*Datensatz*“ bzw. im späteren „*Tupel*“ weder der kleinste noch der größte Nenner für die Definition der relevanten Daten sein kann. Durch eine Zerlegung lässt sich zudem die Frage nach den Fehlern leichter beantworten.

Das kleinste Element, in dem relevante Daten zu finden sind, ist das **Attribut**. Die in ihm enthaltenen Daten bilden den **Attributwert**, bzw. auf die ganze Tabelle betrachtet die **Attributwertmenge**.

Dem Attributwert liegt in der Regel ein **Datentyp** zugrunde. Zum Beispiel können Attribute mit Ziffern den Datentyp *Nummer* haben, aber auch die Definition als *Text* ist möglich. Zusätzlich kann von den Werten ein festgelegtes **Eingabeformat** verlangt werden, zum Beispiel Trennstriche nach je drei Ziffern. Insbesondere bei einem solchen Eingabeformat entspricht der Datentyp trotz der Speicherung von Ziffern meist dem Datentyp *Text*. Diese beiden Eigenschaften gelten generell für das gesamte Attribut, dass heißt der Inhalt der einzelnen Attributwerte ist für die Anwendung dieser Eigenschaften nicht von belang.

Ein Attributwert hat zusätzlich zum Datentyp und Eingabeformat auch eine **Bedeutung**. Diese ist meist nicht automatisch zu erschließen. Wir nehmen dazu an, dass sich die Wertemenge eines Attributes in einer **Domäne** befindet, in denen domänenspezifische Regeln gelten. Anders ausgedrückt beschreibt hier der **Domäentyp** den validen Wertebereich eines Attributwertes gemäß den in der Domäne geltenden Vorgaben. Zum Beispiel muss eine als *Text* gespeicherte Identifikationsnummer anders behandelt werden als ein als *Text* gespeicherter Name, da geringfügige Unterschiede zweier Attributwerte im Fall der Identifikationsnummer eher keine Tippfehler sein dürften, wie dies bei einem Namensfeld vermutet werden könnte. Auch wenn beide Attribute keiner speziellen Formatierung unterliegen und beide vom *Text*-Datentyp sind, kann man hier zur Untersuchung den Domäentyp als *IDNummer* vorgeben. Diese Domäne kann die Regel beinhalten, dass deren Werte als fortlaufende Zahlen behandelt werden sollen und damit ein boolescher Vergleich stattfinden soll statt eines Ähnlichkeitsvergleichs. Zur Veranschaulichung betrachten wir zum Beispiel die Ähnlichkeit zwischen „12“ und „15“. Ausgehend von einer Zeichenfolge könnte man eine Ähnlichkeit 50% annehmen, da immerhin ein Zeichen identisch ist. Als Zahl betrachtet stehen jedoch beide Werte für sich allein.

Datentyp, Eingabeformat und Domäentyp stellen die **Attributeigenschaften** dar und bestimmen, welches **Ähnlichkeitsmaß** gewählt werden muss, das speziell die **Ähnlichkeit** zwischen Attributwerten in deren Attributwertmenge bewertet. Ähnlichkeitsmaße können im Folgenden auch zur Bewertung ganzer Datensätze untereinander definiert werden. **Vergleichsverfahren** benutzen Ähnlichkeitsmaße, um **Ähnlichkeitsfaktoren** zu ermitteln, anhand derer die Gleichheit zweier Werte vorhergesagt werden kann. Wir können zusammenfassen, dass Ähnlichkeitsmaße ein Vorgehen zur Bestimmung der Ähnlichkeit beschreiben, während Ähnlichkeitsfaktoren die Darstellung der Ähnlichkeit als Wert bezeichnen. Diese Begriffe sind mitunter schwer zu unterscheiden. Es wird deutlich, wenn wir zum Beispiel zwei Werte mit der *Identitätsfunktion* „=“ als Ähnlichkeitsmaß bewerten. Dieser boolescher Vergleich (dazu später mehr) kann nun den Ähnlichkeitsfaktor „0“ oder „1“ erzeugen. Wir können den Ähnlichkeitsfaktor jedoch auch als „-1“ und „10“ definieren. Diese abweichende Bestimmung des Ähnlichkeitsfaktors ändert hingegen nicht das Maß, das wir anwenden, um die Werte zu vergleichen.

Weicht ein Attributwert nun von seiner vorgegebenen Form (Datentyp, Eingabeformat und Domäne) ab, kann dies als **Attributwertfehler** bezeichnet werden. Dabei ist zu unterscheiden zwischen **wertunabhängigen** Verstößen, zum Beispiel gegen das Eingabeformat, und **wertabhängigen** Verstößen, wie zum Beispiel der Domäne. Wertunabhängige Verstöße können anhand der vorgegebenen Regeln direkt korrigiert werden. So könnte zum Beispiel das Eingabeformat „5 Zeichen, Buchstaben“ vorgeben, eine Zeichenkette mit mehr als fünf Zeichen zu kürzen. Wertabhängig können Fehler sein, die nicht gegen bestimmte Formatierungsregeln verstoßen, sondern von der Bedeutung der Domäne abweichen. Einfache Verletzungen zum Beispiel gegen den Wertebereich können hierbei wie wertunabhängige Fehler durch geeignete Filter bereinigt werden. Im Folgenden sei hier das Mehrfachauftreten gleichbedeutender Attributwerte näher zu Untersuchen.

Ein wertabhängiger Fehler hat zur Folge, dass Attributwerte in gleichbedeutender aber schriftlich abweichender Form auftreten. Das Auffinden solcher Fehler benötigt ein Grundwissen über den korrekten Inhalt des Attributes. Dieses kann entweder aus den Attributwerten abgeleitet werden oder muss vorgegeben werden. Dabei kann man sich die Eigenschaft zu Nutze machen, dass solche fehlerhaften Attributwerte sich wie **Duplikate** verhalten. Das heißt, dass sich innerhalb der Attributwertmenge Gruppen ähnlicher Werte bilden lassen. So wäre zum Beispiel „Hans“ und „Hnas“ ausreichend ähnlich um ein Duplikat darzustellen. Zur Behebung solcher Fehler kann dann zum Beispiel das **Referenzwertverfahren** angewendet werden. Siehe dazu Abschnitt 2.4.1.

Aus einer Reihe von Attributen ergibt sich schließlich der **Datensatz** oder formaler: Das **Tupel**. Dieses stellt eine Datenzeile einer **Tabelle** oder **Relation** dar. Ein typischer Fehlerfall sind gleichbedeutende Tupel, die sich geringfügig in der Belegung der Attribute unterscheiden. So können im Falle von Abweichungen in den Attributwerten zum Beispiel Attributwertfehler vorliegen. Ähnlich den wertabhängigen Attributwertfehlern ergeben sich Duplikate von Tupeln. Zur Auffindung muss hierbei aber die Ähnlichkeit ganzer Tupel verglichen werden. Die Korrektur dieser erfordert wiederum die Bereinigung der einzelnen Attributwerte zwischen den duplizierten Tupeln. Dazu kommt für Tupel das **Merging** zum Einsatz. Siehe dazu Abschnitt 2.4.2.

Existieren zwischen Tupeln verschiedener oder gleicher Tabellen **Beziehungen**, sind diese Tupel im Verbund zu betrachten. Tupel können in einem solchen Fall auf weitere Tupel verweisen, die als Duplikate existieren. In diesem Fall wollen wir von **duplizierten Verweisen** sprechen. Im Falle von ehemals duplizierten Tupeln kann auch eine alte Beziehung nicht gelöscht worden sein. Dadurch können **verwaiste Schlüssel** zurück bleiben. Beispiel: verweise Tupel A auf C und B auf D und diese seien jeweils Duplikate, so müsste bei Korrektur einer der Verweise angepasst werden.

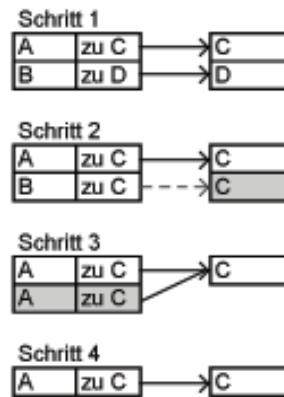


Abbildung 2.1: Nachführende Behebung von duplizierten Tupeln mit Verweisen

Es ist des Weiteren möglich, Tupel und Beziehung als Einheit auf Duplikate zu untersuchen. Dadurch ist es möglich die gesamte Beziehung zwischen Tupeln zu vergleichen. Eine solche Einheit schafft einen neuen Datengegenstand, den wir **Objekt** nennen wollen. Das Objekt wollen wir im Ausblick näher diskutieren, da im Rahmen dieser Arbeit die einfache Beziehungsauflösung zunächst ausreichend ist.

2.1.2 Hierarchien innerhalb der Datengegenstände

Es ist eine Hierarchie zu erkennen, die bei der Anwendung von Bereinigungsabläufen zu beachten ist. Es wurde bereits kurz erwähnt, welche Arten von Fehlern an den Daten auftreten können; dabei verändert die Reihenfolge der Behandlung der Fehler das Endergebnis.

Verwaiste Schlüssel treten in der Regel nach nicht-kaskadierten, d.h. nachgeführten, Löschoperationen auf. Das Beheben solcher Fehler sollte nach Abschluss aller Löschvorgänge erfolgen, damit also als letzte Instanz beim Bereinigen einer Tabelle.

Duplikate sind vor allem in verbundenen Tabellen ein Auslöser für verwaiste Schlüssel. Damit ein Duplikat allerdings zustande kommen kann, muss bereits eine Abweichung auf der Attribut-Ebene vorliegen. Betrachten wir erneut das Beispiel aus Abbildung 2.1 ist hier, vor der Behandlung des Verweises, die Korrektur des Duplikates in der jeweiligen Tabelle mit Hilfe der Attribut-Ebene vorzunehmen. Durch die Korrektur des Duplikates müssen hier nun die Verweise der ersten Tabelle dem neuen Stand der referenzierten Tabelle angepasst werden.

Attributwertfehler sind die häufigste Ursache für Duplikatbildung und treten schon durch geringe Abweichungen bei der Datenübernahme auf. Dabei sei zu bemerken, dass wertabhängige Fehler, die zu Duplikaten von einzelnen Attributwerten führen, nicht zwangsläufig zu einem duplizierten Tupel führen. So brauchen zwei Tupel, die lediglich ein Attribut mit wertabhängigem Fehler gemein haben, kein Duplikat auf der Tupel-Ebene sein. Der Fehler selbst kann aber durchaus ein Duplikat auf Attribut-Ebene darstellen. Dies kann zum Beispiel durch einen Tippfehler eines häufig vorkommenden, typischen Attributwertes für dieses Attribut geschehen.

Für die Datengegenstände ergeben sich somit folgende Hierarchieebenen:

- Attribut-Ebene

- Tupel-Ebene
- Beziehungen bzw. Objekte-Ebene als Erweiterung der Tupel-Ebene

2.2 Ablauf einer Bereinigung

Es bleibt zu klären, wie mit den Fehlern umzugehen ist. Dabei soll hier zunächst das Prinzip der Bereinigung einer einzelnen Tabelle dargestellt werden, an dem sich spätere Erweiterungen orientieren.

Der prinzipielle Ablauf einer Bereinigung lässt sich aus der Hierarchie ableiten. Zunächst sei eine Bereinigung auf Attribut-Ebene und anschließend auf der Tupel-Ebene durchzuführen. Dabei meint Bereinigung wieder die Zusammenfassung von Fehlererkennung und Fehlerkorrektur.

Mit dem Begriff „**hohe Ähnlichkeit**“ soll hier ein Ähnlichkeitsfaktor bezeichnet werden, der ein Paar oder eine Menge als „mit Sicherheit ähnlich“ bezeichnet. Es dazu sei eine Schranke gegeben, die kleiner als der größte erreichbare Ähnlichkeitsfaktor ist. Es soll immer dann von hoher Ähnlichkeit gesprochen werden, wenn diese Schranke vom Faktor überschritten wird. Meist ist diese Schranke als prozentuale Abweichung der Elemente der untersuchten Menge realisiert. Eine niedrig gesetzte Schranke erlaubt folglich, dass auch weniger ähnliche Paare noch mit hoher Ähnlichkeit bewertet werden. Es verringert sich also die „**Präzision**“ der Vorhersage der Ähnlichkeit, während bei hohen Schranken nur sehr ähnliche Paare erkannt werden.

Attribut-Ebene: Wir betrachten für jedes Attribut eines Tupels seine Menge an Attributwerten in der Tabelle. Der Datentyp, das Eingabeformat und der Domämentyp seien uns bekannt. Da Format-, also wertunabhängige, Fehler direkt korrigiert werden können, soll hier bevorzugt die Bereinigung von Fehlern, die einen Ähnlichkeitsvergleich benötigen, betrachtet werden. Aus den Attributeigenschaften leiten wir nun ein geeignetes Ähnlichkeitsmaß ab. Nach der Wahl des Ähnlichkeitsmaßes vergleichen wir jede Kombination von Attributwertpaarungen, die sich aus der Menge bilden lassen und bestimmen deren Ähnlichkeitsfaktoren. Paare, deren Faktoren auf eine hohe Ähnlichkeit schließen lassen, bestenfalls so, dass man von einer Gleichheit ausgehen kann, werden nun mit einem Referenzwert abgeglichen. Anhand eines solchen Wertes wird entschieden, welche Eingabe des Attributwertes als die korrekte Eingabe anzusehen ist. Es stellt sich der folgende Plan dar:

```
//Fehlererkennung:
Für jedes Attribut
  Wähle Ähnlichkeitsmaß
  fertig:=falsch
  Solange noch nicht fertig
    fertig:=wahr
    Für jedes Wertepaar der Attributwertmenge
      Erzeuge Ähnlichkeitsfaktor
      Wenn Faktor hoher Ähnlichkeit aber nicht identisch entspricht
        //Fehlerkorrektur
        fertig:=falsch
```

Ermittle Referenzwert nach Abschnitt 2.4.1
 Wende Korrektur durch Referenzwert an

Auflistung 2.1: Bereinigung wertabhängiger Fehler der Attribut-Ebene

Der in Auflistung 2.1 dargestellte Plan zeigt ein einfaches Vorgehen zur Erkennung von Duplikaten. Dabei wird jeder Wert mit jedem weiteren Wert verglichen. Stellt sich dabei eine hohe Ähnlichkeit ein, so wird das Paar korrigiert. Nach dem Korrigieren von später behandelten Paaren, können hohe Ähnlichkeiten mit früheren Werten erzeugt worden sein. Daher wird der gesamte Vorgang solange wiederholt, bis nur noch identische oder unähnliche Paarungen übrig sind.

Tupel-Ebene: Um Tupel miteinander zu vergleichen, müssen wir die einzelnen Attribute der Tupel miteinander vergleichen, und daraus einen Ähnlichkeitsfaktor für das Tupelpaar ableiten. Die Bereinigung auf Tupel-Ebene setzt sich also zusammen aus einem Vergleich auf der Attribut-Ebene, einem daraus folgernden Vergleich auf der Tupel-Ebene und der Fehlerkorrektur. Im Falle der Fehlererkennung ergibt sich nun der Bedarf eines Ähnlichkeitsmaßes, das Tupel bewertet. Dieses muss auf dem Vergleich auf der Attribut-Ebene aufbauen. Ein solches Maß kann zum Beispiel die Aufsummierung der Ähnlichkeitsfaktoren der Attribute sein. Der Ähnlichkeitsfaktor wäre in diesem Beispiel das Ergebnis dieser Summierung. Bei der Fehlerkorrektur müssen nun unterschiedliche Attributwerte der Duplikate zu einem Tupel kombiniert werden und, im Gegensatz zur Attribut-Ebene, das duplizierte Tupel entfernt werden. Diese Art der Fehlerkorrektur heiße in dieser Arbeit **Merging**, was in Abschnitt 2.4.2 vorgestellt wird.

```
//Fehlererkennung:
(*)
fertig:=falsch
Solange noch nicht fertig
  fertig:=wahr
  Für jedes Paar der Tupelmenge
    Für jedes Attribut
      Wähle Ähnlichkeitsmaß
      Erzeuge Ähnlichkeitsfaktor für das erhaltene Attributwertpaar
      Verrechne diesen Faktor mit dem Ähnlichkeitsfaktor des Tupelpaares
    Wenn Ähnlichkeitsfaktor des Tupelpaares hoher Ähnlichkeit entspricht
      //Fehlerkorrektur:
      Wende Merging an
      Entferne eines der Tupel (eventuell Bevorzugung vorgegeben)
      fertig:=falsch
(**)
```

Auflistung 2.2: Bereinigung auf Tupel-Ebene

Für Tabellen mit Verweisen auf Tupel fremder Tabellen ist es nötig, zunächst dem Verweis zu folgen und die fremde Tabelle zu bereinigen, bevor die referenzierende Tabelle

bereinigt wird. Dies wird deutlich, wenn wir zu einer Tabelle „*Bücher*“ die Tabelle „*Ausleiher*“ referenzieren. Wir verwenden wieder das Beispiel aus Abbildung 2.1. Seien nun in der Tabelle „*Bücher*“ die Bücher A und B Duplikate. Buch A verweist auf Ausleiher C und B auf Ausleiher D, die beide erneut ein Duplikat seien. Nehmen wir an, dass dieser Verweis als untrennbare Zusatzinformation zu verstehen ist. Würde zunächst „*Bücher*“ bereinigt, würde das Duplikat von A und B nicht auffallen, da beide unterschiedliche Zusatzinformationen beinhalten. Anschließend würde nur das Duplikat von C und D erkannt. Beginnt man hingegen mit der referenzierten Tabelle „*Ausleiher*“, werden sowohl C und D als auch anschließend A und B erkannt. Es ergeben sich also folgende Einfügungen zur Auflistung 2.2:

```
//Fehlererkennung
(*) Stelle Tabelle zurück und Bereinige eventuelle referenzierte Tabellen
//Fehlerkorrektur
(**) Korrigiere verwaiste Schlüssel in referenzierender Tabelle
```

Auflistung 2.3: Einfügungen zur Bereinigung von Verweisen

Betrachtet man die Ablaufpläne der Attribut- und Tupel-Ebene, stellt man fest, dass beide eine gewisse Ähnlichkeit aufweisen. So ist beiden gemeinsam, dass die Attributwerte bewertet werden und dass Paare gebildet werden, die iterativ ausgewertet werden. Nun ist dieser einfache Ansatz zwar sicher funktional, bei hohem Datenvorkommen aber lauffzeittechnisch sicher nicht wünschenswert. Es stellt sich die Frage, ob sich Verfahren finden lassen, die diese Iterationen vermeiden oder vermindern. So kann man beispielsweise Paare mit hoher Ähnlichkeit gruppieren. Mit zusätzlichen Verbesserungen findet sich dieser Vorschlag in der Literatur unter dem Begriff **Clustering**. Für solche Verbesserungen siehe auch [Bra07]. Betrachten wir Attribut- und Tupel-Ebene noch einmal in Hinblick auf diese Paarbildung, fällt auf, dass sowohl das Auffinden von duplizierten Tupeln als auch von Attributwerten einen identischen Vorgang durchführen. In der Tat können wir erkennen, dass das Auffinden von ähnlichen Attributwerten sich verhält wie das Auffinden von duplizierten Tupeln, wenn wir das Tupel auf ein einziges Attribut reduzieren. Wir können also annehmen, dass sich optimierende Techniken, wie sie von Verfahren zur Duplikaterkennung zumeist für Tupel eingesetzt werden, auch auf Attribut-Ebene als Derivat verwenden lassen.

Offensichtlich werden zur Fehlererkennung nun Verfahren benötigt, die Ähnlichkeitsmaße bilden und Ähnlichkeitsfaktoren erstellen können, sowie in der Lage sind, diese lauffzeittechnisch zu optimieren. Im folgenden Abschnitt werden einige bekannte Verfahren genannt, von denen wir später spezielle Ausgewählte genauer betrachten werden.

2.3 Verfahren zur Fehlererkennung

Zur Ausführung des in Abschnitt 2.2 grob dargestellten Ablaufs bedarf es Verfahren, die die Ähnlichkeitsfaktoren bestimmen und die Vergleichsoperation auf hohe Ähnlichkeit ausführen können. Vor allem bei nicht-boolschen Vergleichen, also Vergleichen die nicht direkt mit „ja“ oder „nein“ beantwortet werden können, ist die zuvor angesprochene Präzision der Verfahren ein entscheidender Faktor. Dabei darf allerdings nicht vergessen

werden, dass bei einem erheblichen Datenumfang der Rechenaufwand von aufwändigen Verfahren sich exponentiell niederschlagen kann. In der Literatur finden sich viele Verfahren, die solche Vergleichsoperationen umsetzen. Eine kompakte Übersicht bietet hierzu zum Beispiel der Artikel [EIV07].

In diesem Abschnitt sollen einige Verfahren genannt werden, die auf den in 2.1.2 gezeigten Ebenen angewendet werden können. Dabei sei anzumerken, dass es sich bei den hier vorgestellten Verfahren auf Attribut-Ebene um Ähnlichkeitsmaße handelt, während die Tupel-Ebene sich mit der Erkennung von Duplikaten mit laufzeitoptimierten Verfahren befasst. Wie wir aus Abschnitt 2.2 sehen konnten, hat dies den Grund, dass auch auf Tupel-Ebene im Endeffekt die Ähnlichkeitsmaße der Attribut-Ebene zum Einsatz kommen und auf Attribut-Ebene die Vergleichsoperationen der Tupel-Ebene anwendbar sind.

2.3.1 Attribut-Ebene

Vergleichsoperationen auf der Attribut-Ebene beschäftigen sich im Allgemeinen mit der Frage wie ähnlich sich zwei Zeichenketten sind. Das Ziel ist, einen Ähnlichkeitsfaktor zu ermitteln, der bestenfalls darauf hinweist, dass eine Zeichenkette aus einer anderen eindeutig hervorgeht. Damit sollen Tippfehler und geringfügige Abweichungen behandelt werden können. Dazu gibt es primär den zeichen- und den lautbasierenden Ansatz um Ähnlichkeitsmaße zu konstruieren. Numerische Vergleichsoperationen und Maße seien hiermit als einfache Derivationen erwähnt.

Der lautbasierende Ansatz hat als Ziel, Wörter mit gleicher Phonetik aber stark abweichender Schreibweise ermitteln zu können. Da dies für diese Arbeit nicht notwendig ist, sei hiermit als Beispiel auf das **Soundex** Verfahren verwiesen was in [HM02] zur Anwendung kommt.

Häufige zeichenbasierende Ansätze sind zum Beispiel die Maße:

Edit-Distanz: Die Edit-Distanz stellt ein einfaches Verfahren dar, um die „Distanz“ und daraus abgeleitet die Ähnlichkeit zweier Zeichenketten zu bestimmen (vgl. [Lev66]). Dabei wird bestimmt, wie viele Ersetzungsoperationen auf einen Ausdruck angewendet werden müssen, um den jeweils anderen Ausdruck zu erhalten. Berücksichtigt wird jedoch nicht welche Zeichen im Ausdruck gesamt verwendet wurden. So ist die Distanz von „Mainz“ und „Heinz“ genauso groß wie „Heinz“ zu „Heniz“. Dass es sich bei letzterem um eine höhere Ähnlichkeit durch einen vermutlichen Tippfehler handelt, ist nicht unmittelbar zu entnehmen. Es bedarf dazu einer Erweiterung des ursprünglichen Verfahrens.

Q-Grams: Beim Q-Gram-Verfahren werden Zeichenketten in Teilketten (Q-Grams) der Länge q aufgeteilt. Die Theorie ist, dass wenn Zeichenketten ähnlich sind, sich viele solcher Q-Grams überlappen. Der Ähnlichkeitsfaktor zweier Zeichenketten ermittelt beispielsweise sich aus der Anzahl solcher Überlappungen zur Gesamtzahl der Q-Grams. Eine umfangreiche Erweiterung als vollständige Vergleichsoperation ist dem Artikel [MB06] zu entnehmen. Dieser beschäftigt sich mit der Verwendung von Q-Grams zur Erkennung von ähnlichen Attributwerten in Attributwertmengen, schließt also neben dem Ähnlichkeitsmaß auch eine Anwendung in hier spezialisierten optimierten Verfahrensweisen für Mengen ein, zum Beispiel Cluster-Bildung.

Field-Similarity: Dies ist ein einfaches Verfahren, das die Abweichung zweier Zeichenketten berechnet, indem es jede Zeichenkette in seine Zeichen zerlegt und ein wiederholtes Auftreten dieser vermerkt. Die so erhaltenen Mengen an Zeichen-Auftritts-Elementen

werden geschnitten. Das Ergebnis des Schnitts, ins Verhältnis zur Zeichenanzahl der Ausgangszeichenketten gesetzt, ergibt dann einen Ähnlichkeitsfaktor. Ein detaillierter Einblick findet sich im Kapitel 3.3; dort wird auch die Verwandtschaft zu Q-Grams deutlich. Im Gegensatz zur Edit-Distanz ist bezieht dieses Verfahren den Wertevorrat bei der Bewertung mit ein.

Als Erweiterung kann man Token-Basierende Ähnlichkeitsmaße sehen. Ein Beispiel sei das **Atomic String** Verfahren nach [ME96]. Dieses Verfahren vergleicht atomare Teilzeichenketten auf Gleichheit und ist damit in der Lage „Hans Muster“ und „Muster Hans“ als identisch zu erkennen. Eine Kombination mit einem zeichenbasierenden Ähnlichkeitsmaß wäre denkbar (siehe dazu auch Abschnitt 4.7.4).

2.3.2 Tupel-Ebene

Auf der Tupel-Ebene wird versucht Duplikate zu erkennen. Wie bereits gezeigt, können mit entsprechendem Ähnlichkeitsmaß auf Attribut-Ebene, mit diesen Verfahren auch Duplikate in Attributwertmengen erkannt werden, also folglich auch wertabhängige Fehler. Die einfachste Vergleichsoperation ist der boolsche Vergleich. Dabei wird ein als Schlüssel fungierendes Attribut, das für jedes Tupel einen individuellen Wert beinhaltet, auf Übereinstimmung geprüft.

Aufwändiger wird die Duplikaterkennung, wenn über die Ähnlichkeit von Tupeln entschieden werden muss. Nach einer ersten groben Rasterung kann man die Ansätze zu verschiedenen Verfahren einteilen in lernende und algorithmenbasierende Ansätze. Den lernenden Ansätzen liegt im Großen eine Benutzerinteraktion zugrunde, nach der ein Muster gebildet wird, anhand dessen sich das Ähnlichkeitsmaß bei der Entscheidungsfindung orientiert. Da für diese Arbeit möglichst wenig Interaktion gewünscht wird, sei auf diese Ansätze hier nur verwiesen. Eine Übersicht findet sich erneut im Artikel [EIV07].

Bei den algorithmenbasierenden Ansätzen interessieren vor allem die Laufzeiten. Ein einfacher Vergleichsalgorithmus wurde bereits in Abschnitt 2.2 gezeigt. Die Laufzeit eines solchen Algorithmus ist verständlicherweise sehr hoch. Die häufigsten vorzufindenden Verfahren lassen sich einordnen in fensterbasierend und clusterbasierend.

Bei Fensterbasierenden Verfahren werden die Tupel nach einem Schlüssel sortiert, der so berechnet wird, dass ähnliche Tupel nah beieinander liegen. Anschließend wird mit einer festen Fenstergröße über die Datensätze gegangen und diese jeweils innerhalb des Fensters untereinander verglichen. Ein häufig erwähnter Vertreter ist die **Sorted Neighborhood Method** (SNM) [HS98].

Clusterbasierende Verfahren gehen von einer angenommenen Transitivität von Ähnlichkeiten aus. Das heißt: Wenn A ähnlich zu B ist und B ähnlich zu C, so ist A in gewissen Schranken ähnlich zu C. Unterteilt man den Datenbestand nun in Abschnitte, also Cluster, in denen die Ähnlichkeit nur zu einem stellvertretendem Tupel innerhalb des Clusters berechnet wird, braucht zum näheren Vergleich nur noch dieses stellvertretende Tupel genutzt werden. Von dem oben erwähnten SNM Verfahren gibt es als Ergänzung dazu auch das **Clustering-SNM** [HS95]. Dabei wird nach dem Sortieren der Datenbestand in kleine Cluster unterteilt und in denen zunächst unabhängig SNM angewendet. Dabei wird davon ausgegangen, dass bei der vorhergehenden Sortierung alle ähnlichen Tupel bereits nahe beieinander liegen. Es werden die Vergleichsoperationen an den Clusterübergängen vermieden, die beim normalen SNM vorkommen.

Kombiniert man diese Technik noch mit dem Aufspalten in Ähnlichkeitspartitionen, gelangt man zum Verfahren **PC-Filter** aus [ZLBL04], das in Kapitel 3.5 vorgestellt wird.

2.3.3 Objekt-Ebene

Würde man die Objekt-Ebene realisieren, so unterscheidet sie sich von der Tupel-Ebene hauptsächlich davon, dass zur Erkennung von Duplikaten enthaltene Verweise auf andere Tabellen mit berücksichtigt werden. Dazu kann eine Beziehung mit einem Ähnlichkeitsmaß bewertet werden und dessen Faktor dem Objekt mit zugerechnet werden. Anschließend kann eines der oben genannten Duplikatverfahren angewandt werden.

2.4 Verfahren zur Fehlerkorrektur

In diesem Abschnitt wird verdeutlicht, welche Optionen es zur Fehlerkorrektur auf Attribut-Ebene mit Hilfe eines Referenzwertes und auf Tupel-Ebene durch Merging gibt.

2.4.1 Korrektur mit Referenzwert

Wurde durch die Fehlererkennung eine Menge von Attributwerten gefunden, die eine hohe Ähnlichkeit aufweisen (also einen Cluster bilden), so stellt sich die Frage, welcher dieser Attributwerte nun der zur gewünschten Darstellung passende Wert ist. Die eigentliche Korrektur besteht aus dem einfachen Ersetzen aller nicht passenden Werte. Dazu kann man mehrere Ansätze verfolgen.

Zunächst besteht die Möglichkeit, schon vor der Bereinigung Referenzwerte vorzugeben, von denen der Passende zur Laufzeit anhand der Ähnlichkeit zu den Attributwerten bestimmt werden kann. Es wird also in einem Lexikon von Referenzwerten nach einem passenden gesucht. Ein Anwendungsfall zur effektiven Anwendung vorgegebener Referenzwerte wäre zum Beispiel das automatische Ersetzen von Abkürzungen gegen die ausgeschriebenen Begriffe. Der umgekehrte Fall wäre, dass der Benutzer zu der Menge während der Laufzeit den passenden Referenzwert eingibt.

Der Referenzwert kann automatisch ermittelt werden, indem der am häufigsten vorkommende Attributwert ermittelt wird. Dessen Auftretshäufigkeit sollte dabei eine zuvor gesetzte Schranke überschreiten. Dies ist nötig, um sicherzustellen, dass der Wert definitiv der dominanteste Wert der Menge ist, so dass eine Fehlentscheidung für die Menge vermieden wird. Es kann ebenfalls die Ähnlichkeit zwischen dem so ermittelten Wert und den übrigen Attributwerten verglichen werden, um Fehlentscheidungen in Einzelfällen zu vermeiden.

Kombinationen dieser Ansätze sind möglich. So kann der Auswertung vorgegebener Referenzwerte die automatische Ermittlung folgen. Können die jeweils vorgestellten automatischen Verfahren keinen Referenzwert ermitteln, zum Beispiel durch Nichterreichen einer Mindesthäufigkeit, muss der Benutzer entscheiden.

2.4.2 Merging

Auf der Tupel-Ebene kommt man schnell zu der Frage, welches Tupel denn das Duplikat und welches das Original ist. In manchen Fällen gibt es darauf keine explizite Antwort. So hat Tupel A eine abweichende Belegung vom als Duplikat erkannten Tupel B bei gleich hohem Informationsgehalt. Mit einer größeren Anzahl duplizierter Tupel können auch sporadisch gänzlich abweichende Attributwerte auftreten. Dieses Problem so weit wie möglich automatisch zu beheben ist Aufgabe des Merging.

Das Merging behandelt ähnlich zum Referenzwertverfahren eine Menge von Tupeln mit hoher Ähnlichkeit (also ein Cluster von duplizierten Tupeln). Es wird für alle Tupel die Attribute durchlaufen. Es ergeben sich also Mengen an Attributwerten. Dabei können folgende Fälle auftreten:

1. alle Attributwerte identisch.
2. nur ein nicht leerer Attributwert.
3. Attributwerte unterscheiden sich.

Das Vorgehen für die einzelnen Fälle ließe sich wie folgt beschreiben:

1. Attribut wird übersprungen, da keine Aktion nötig.
2. alle Attributwerte werden auf den einen Wert gesetzt.
3. zur Ermittlung des Attributwertes wird die Menge an das Referenzwertverfahren übergeben.

Es erscheint praktikabel unter Verwendung von automatisch ermittelnden Referenzwertverfahren folgende Vorgaben zu beachten:

- wenn alle Attributwerte identisch, sind ist der automatisch ermittelte Referenzwert eben solcher.
- leere Attributwerte werden bei der automatischen Ermittlung übersprungen.
- leere Attributwerte werden auf den Referenzwert gesetzt.

Werden diese Vorgaben erfüllt, kann hierbei für jeden Fall das Referenzwertverfahren aufgerufen werden.

Nach Durchlaufen der Attribute wird ein Tupel als Original bestimmt und der Rest entfernt. Anschließend sind die Verweise auf die entfernten Tupel auf das Originaltupel zu ändern.

Beim Merging gibt es zwei mögliche Benutzerinteraktionen. Zum Einen kann im Sinne des Referenzwertverfahrens eine Interaktion nötig werden. Zum Anderen besteht die Möglichkeit, dass Tupel in den Cluster eingeordnet wurden, die kein richtiges Duplikat zu den anderen Tupeln sind. Diese sollten durch eine Benutzerinteraktion aus dem Cluster entfernt werden. Dazu ein Beispiel:

Behandelt wird eine Tabelle mit Personendaten.

Duplikate wurden anhand von Nachnamen und Vornamen erkannt.

Die als Duplikate erkannten Personen haben unterschiedliche Adressen und Telefonnummern.

Die Personen stellen keine Duplikate dar.

Beispiel 2.1: Clusterkorrektur bei unzureichender Duplikaterkennung

Das Merging geht auch hier, gemäß der Definition, davon aus, dass es sich bei der Menge an Tupeln um Duplikate handelt. Die Behandlung erfolgt folglich wie oben beschrieben. Nun treten bei den Adressen und Telefonnummern eventuell eine Interaktionsanfrage aus dem Referenzwertverfahren auf. Hier muss der Benutzer auch entscheiden, ob das betroffene Tupel überhaupt zum Cluster gehört, also wirklich ein Duplikat ist. Dass die hier angewendete Duplikaterkennung nicht ausreichend war, erkennt das Verfahren nicht ohne Interaktion.

Eine automatische Erkennung von schlecht gewählten Tupeln kann anhand einer großen Anzahl leerer Attributwerte erfolgen. So zum Beispiel, wenn vergleichsweise leere Tupel sich im Cluster befinden, dessen Unterscheidungspunkt zwar ähnlich, aber gänzlich anders bedeutend ist. Dieser Sonderfall tritt beispielsweise bei referenzierten Tupeln auf, die Zusatzinformationen zu verweisenden Tupeln beinhalten. Hier wurde das Tupel anhand eines Attributes eingeordnet, das zwar hohe Ähnlichkeit aufweist, nicht aber auf Objekt-Ebene verträglich ist. Zum Beispiel wird geprüft auf das Attribut „Name“, und erkannt wird „Hans“ und „Hans“. Ersteres Tupel enthält aber leere Attributwerte, während das zweite Tupel noch eine Adressinformation beinhaltet. Nach dem beschriebenen Vorgehen würden alle Attribute des ersten Tupels ohne Nachfrage angepasst. Auf diese Tupel verweisende Tupel können hingegen völlig unterschiedlich sein. Da auf Tupel-Ebene nicht die Beziehung bewertet wird, kann als Schutzmaßnahme bei einer solchen Erkennung eine Interaktion eingefordert werden.

2.5 Bibliographischer Datenbestand des DBIS

Die in dieser Arbeit vorgestellten Werkzeuge sollen auf die bibliographische Datenbank vom Fachgebiet Datenbanken und Informationssysteme angewendet werden. Dazu stellen wir hier zunächst den Aufbau der Datenbank vor und erläutern die Besonderheiten, an denen die Werkzeuge angepasst werden müssen.

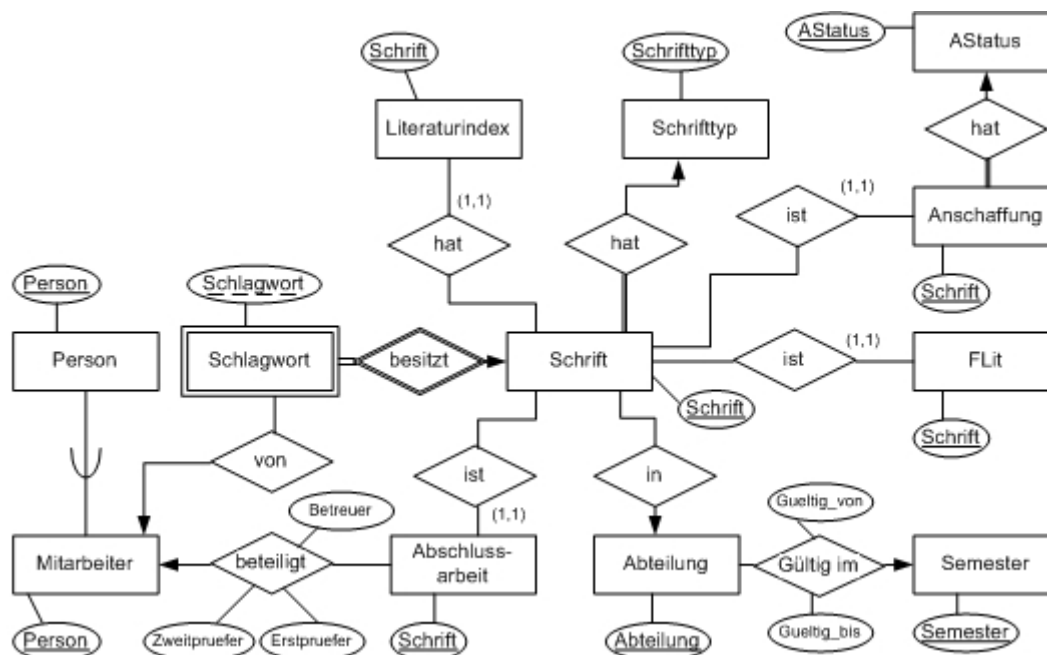


Abbildung 2.2: Entity-Relationship-Model

Aus Gründen der Übersichtlichkeit wurden die Attribute der Tabellen nicht in die Abbildung 2.2 mit aufgenommen. Die relevanten Attribute werden im Folgenden erläutert. Dazu wählen wir zunächst die am häufigsten von Fehlern betroffenen Tabellen aus dem Schema:

Die Einträge der Tabelle **Schrift** stellen die primären Einträge für Schriftstücke dar. Ein Schriftstück muss dabei mindestens einen *Titel* und einen *Schrifttyp* haben. Die relevanten Attribute und die typischen Fehlerfälle dieser Tabelle sind:

- *Autor*: Formatabweichung („Nachname1, Vorname1 / Nachname2, Vorname2“ etc.), abweichende Eingabereihenfolge der Co-Autoren.
- *Titel*: Abweichende Titelbezeichnung.

Die Einträge der Tabelle **FLit** stellen weitere Informationen zu einem Schriftstück bereit. Darunter auch die *ISBN*, *Verlag* und *Erscheinungsjahr* sowie *Ort*.

- *ISBN*: Uneinheitliches Format (10 oder 13 stellig, Trennzeichen).
- *Verlag*: Mehrfachnennung von Verlagen durch geringe Unterschiede und Abkürzungen.
- *Ort*: Mehrfachnennung von Orten durch geringe Unterschiede und Abkürzungen.
- *Sprache*: Vereinzelt Nullwerte (zwar zulässig, aber meist hat ein Schriftstück eine Sprache).

Die Tabelle **Schlagwort** stellt zu Schrift Schlagwörter als Suchbegriffe bereit. Aufgrund der Schlüsselbeziehung sind Schriftstück und Schlagwort immer eindeutig. Das heißt, dass es zu einem Schriftstück kein doppeltes Schlagwort gibt. Das *Schlagwort* Attribut jedoch kann unterschiedliche gleichbedeutende Eingaben enthalten. Folgendes Beispiel ist während der Erstellung dieser Arbeit in der Datenbank zu finden:

Schrift: 12896, Schlagwort: „view update“

Schrift: 12899, Schlagwort: „view upadtes“

Beispiel 2.2: Dupliziertes Schlagwort

Die Tabelle **Person** ist ein Personenverzeichnis verschiedener Personen der Universität. Als Teilmenge davon werden in der Tabelle **Mitarbeiter** die Mitarbeiter des Fachgebiets aufgenommen. Die Personeneinträge können Duplikate enthalten, die durch Abweichungen von folgenden Attributen unterschieden werden können:

- *Titel*
- *EMail*
- *Homepage*

Es bleibt festzuhalten, dass wir **Duplikatbereiniger** auf Attribut- und Tupel-Ebene benötigen, die auch in der Lage sind, Tippfehler und andere geringe Abweichungen zu erkennen. **Ähnlichkeitsmaße** müssen mit Zeichenketten umgehen können, die sich aus mehreren Wörtern zusammensetzen. Wir benötigen **Formatbereiniger**, die speziell für ISBN-Formatierung und das Einhalten des Autoren-Formats optimiert sind. Und es wird ein Werkzeug gebraucht, mit dem wir **Nullwerte** auffinden können.

Bei der Korrektur der gefundenen Fehler auf der Tupel-Ebene ist darauf zu achten, dass Verweise auf die Tabelle **Schrift** korrigiert werden, sollte in dieser ein Schriftstück gelöscht worden sein. Ebenso sind alle Tabellen mit Foreign-Keys zu behandeln, wie zum Beispiel Tabelle **Abschlussarbeit** zu **Mitarbeiter**.

Kapitel 3

Ausgewählte Verfahren

In diesem Kapitel werden aus der Literatur ausgewählte Verfahren im Detail besprochen, die in dieser Arbeit zur Anwendung kommen werden. Dabei handelt es sich um Verfahren zum Erzeugen eines Ähnlichkeitsmaßes sowie auch eines Ähnlichkeitsfaktors für Zeichenketten und dem Erkennen von Tupel-Duplikaten.

3.1 Überblick

Wir haben in Bezug auf Abschnitt 2.2 drei Probleme zu lösen. Zum Einen müssen wir Ähnlichkeiten zwischen Attributwerten bestimmen können. Dann müssen wir eine Menge von Attributwerten effektiv miteinander vergleichen und dieses zum Dritten auf Tupel ausweiten. Durch die Auszeichnung einer Attributwertmenge als Tupel mit nur einem Attribut entfällt dieser letzte Schritt. Wir brauchen also nur ein Verfahren zu wählen, dessen Duplikaterkennung vom eigentlichen Wert des Attributes unabhängig gestaltet werden kann.

Dazu wollen wir das Q-Gram Verfahren zur Ähnlichkeitsfaktorbildung vorstellen. Der Vorteil dieses Verfahrens ist die Eignung zur Realisierung auf Datenbankebene, da das Prinzip des Schneidens zweier Listen bereits in optimierter Form vorhanden ist. Dazu werden wir zeigen, dass hier insbesondere Tippfehler leicht von gänzlich anderen Worten unterscheidbar sind trotz gleichen Zeichenvorrats. Zudem werden wir zeigen, dass das Field-Similarity Verfahren eine Ableitung des Q-Gram-Verfahrens darstellt und sich aufgrund der Einfachheit leicht durch Kombinationen verbessern lässt. Als Ähnlichkeitsmaß für Zeichenketten wollen wir die Verfahren so festlegen, dass zwei Zeichenketten eingegeben werden und deren Ähnlichkeitsfaktor im Intervall zwischen 0 und 1 ausgegeben wird. Dies ermöglicht die direkte Bestimmung des Ähnlichkeitsfaktors zweier beliebiger Zeichenketten und erlaubt eine Verwendung im späteren PC-Filter Verfahren zur Bestimmung der Ähnlichkeitsfaktoren auf Attribut-Ebene.

Das PC-Filter Verfahren wird in der Literatur als robuster Duplikaterkenner auf Tupel-Ebene vorgestellt. Das Prinzip dahinter ist eine Kombination aus Ähnlichkeitsvergleich und abgewandelter Clustertechnik. Dabei kann das Verfahren zur Ähnlichkeitsfaktorbildung für die Attribut-Ebene sehr leicht ausgetauscht werden, solange es sich an die später vorgestellten Regeln hält. Dieses Verfahren teilt den Datenbestand in Partitionen ein, die wie Cluster verwaltet werden und ermöglicht ein zeiteffektives Auffinden von Duplikaten

auch in Nachbarpartitionen. Damit ist es den Verfahren die nur *Sliding-Window* einsetzen überlegen. Beim *Sliding-Window* wird eine Menge an Tupeln durchlaufen und eine feste Anzahl an aufeinander folgenden Tupeln miteinander verglichen. Es erfolgt also kein Vergleich weit entfernter Tupel.

Wir haben dieses Verfahren der *Sorted Neighborhood Method* vorgezogen, da es zum Einen für einen großen Datenbestand in der Lage ist, durch geringen Mehraufwand auch weiter entfernte Daten zu vergleichen. Zudem ist es durch die Abtrennung des Ähnlichkeitsvergleiches mit der eigentlichen Duplikaterkennung in der Lage mit austauschbaren Ähnlichkeitsmaßen implementiert zu werden. Des Weiteren läuft es in überschaubaren Phasen ab, so dass eine externe Steuerung möglich wird.

Dem Verfahren wird dabei eine Menge an Tupeln vorgegeben, sowie die Anzahl an Partitionen, die Größe des Partitionsrandbereiches und der gewünschte Genauigkeitsgrad. Die Größe der Partionen und der Partitionsrandbereich beeinflussen dabei die Anzahl der Iterationen über die Partitionen. Eine Erhöhung des Randbereichs ermöglicht eine genauere Erkennung von Duplikaten zu Tupeln in verschiedenen Partitionen. Durch die steigende Anzahl der Vergleiche verlängert sich aber auch die Ausführzeit. Die Ausgabe des Verfahrens sind Cluster mit Tupeln hoher Ähnlichkeit.

PC-Filter erfüllt die oben genannte Vorgabe für einen Duplikaterkenner für Attributwerte und ist zudem in der Lage, auch entfernte Tupel bzw. Attributwerte zu vergleichen. So ist es mit PC-Filter möglich mit entsprechenden Ähnlichkeitsmaß „Hans Müller“ und „Müller Hans“ als Duplikat erkennbar zu machen.

3.2 Q-Grams

Das Q-Gram-Verfahren [Ukk92] zerteilt die Eingangszeichenketten in Teilstrings (Q-Grams) der Länge q . Dabei werden die Eingangszeichenketten durch ein sich verschiebendes Fenster dieser Größe q aufgeteilt. Um die Anfangs- und Endbuchstaben einzeln auswerten zu können, ist es für $q > 1$ nötig, vor und hinter der jeweiligen Eingangszeichenkette noch spezielle Steuerzeichen zu setzen, die den Anfang sowie das Ende markieren und in der Zeichenkette nicht vorkommen. Dies können beispielsweise die Zeichen „\$“ und „#“ sein. Jedes Auftreten eines Teilstrings wird mit einer Auftrittsnnummer versehen. Diese gibt an, wie oft der Teilstring bereits aufgetreten ist.

Als Beispiel sei die Eingangszeichenkette „GeorGe“ gegeben. Als q wählen wir 2. Die resultierende Menge ist $\{\#G1, Ge1, eo1, or1, rg1, Ge2, e$1\}$.

Diese Mengen, die sich aus zwei Eingangszeichenketten ergeben, werden anschließend geschnitten. Die Schnittmenge enthält alle überlappenden Teilstrings der Länge q der beiden Eingangszeichenketten.

Das Ähnlichkeitsmaß ist also das Überlappen, bzw. die Anzahl der Überlappungen der Q-Grams zwischen den Zeichenketten.

Den hier vorgestellten Ähnlichkeitsfaktor erhalten wir durch die Anzahl der Überlappungen in Relation zu der Gesamtzahl der Elemente des Verbundes der Eingangsmengen.

- $Z_1 = \text{“Hanna“}$

$$\begin{aligned}
Z_2 &= \text{“Hanan“} \\
T_1 &= \{\#H1, Ha1, an1, nn1, na1, a\$1\} \\
T_2 &= \{\#H1, Ha1, an1, na1, an2, n\$1\} \\
T_1 \cap T_2 &= \{\#H1, Ha1, na1, an1\} \\
T_1 \cup T_2 &= \{\#H1, Ha1, an1, nn1, na1, a\$1, an2, n\$1\} \\
sim(T_{Z_1}, T_{Z_2}) &= \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} = \frac{4}{8} = \frac{1}{2} = 0,5
\end{aligned}$$

Beispiel 3.1: Zeichendreher mit $q=2$

- $$\begin{aligned}
Z_1 &= \text{“Henne“} \\
Z_2 &= \text{“Hanna“} \\
T_1 &= \{\#H1, He1, en1, nn1, ne1, e\$1\} \\
T_2 &= \{\#H1, Ha1, an1, nn1, na1, a\$1\} \\
T_1 \cap T_2 &= \{\#H1, nn1\} \\
T_1 \cup T_2 &= \{\#H1, Ha1, an1, nn1, na1, a\$1, He1, en1, ne1, e\$1\} \\
sim(T_{Z_1}, T_{Z_2}) &= \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} = \frac{2}{10} = \frac{1}{5} = 0,2
\end{aligned}$$

Beispiel 3.2: Austausch von zwei Zeichen mit $q=2$

- $$\begin{aligned}
Z_1 &= \text{“Mainz“} \\
Z_2 &= \text{“Heinz“} \\
T_1 &= \{\#M1, Ma1, ai1, in1, nz1, z\$1\} \\
T_2 &= \{\#H1, He1, ei1, in1, nz1, z\$1\} \\
T_1 \cap T_2 &= \{in1, nz1, z\$1\} \\
T_1 \cup T_2 &= \{\#M1, Ma1, ai1, in1, nz1, z\$1, \#H1, He1, ei1\} \\
sim(T_{Z_1}, T_{Z_2}) &= \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} = \frac{3}{9} = \frac{1}{3} \simeq 0,3
\end{aligned}$$

Beispiel 3.3: unähnliche Zeichenfolge mit $q=2$

- $$\begin{aligned}
T_1 &= \{\#\#M1, \#Ma1, Mai1, ain1, inz1, nz\$1, z\$\$1\} \\
T_2 &= \{\#\#H1, \#He1, Hei1, ein1, inz1, nz\$1, z\$\$1\} \\
T_1 \cap T_2 &= \{inz1, nz\$1, z\$\$1\} \\
T_1 \cup T_2 &= \\
&\{\#\#M1, \#Ma1, Mai1, ain1, inz1, nz\$1, z\$\$1, \#\#H1, \#He1, Hei1, ein1\} \\
sim(T_{Z_1}, T_{Z_2}) &= \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} = \frac{3}{11} \simeq 0,27 < \frac{1}{3}
\end{aligned}$$

Beispiel 3.4: unähnliche Zeichenfolge mit $q=3$

Wie wir an diesen Beispielen sehen können, reagiert dieses Verfahren empfindlich auf die Position, an denen Zeichenketten sich unterscheiden. Nahe beieinander liegende Unterschiede ergeben eine höhere Ähnlichkeit als Verstreute. Um speziell Zeichendreher aufzufinden zu können, würde es sich anbieten, das q auf 1 zu reduzieren. So führe dies im „Hanna“ Beispiel sogar auf einen Faktor von 1.

Ein Verfahren der Literatur, dass sich mit der Anwendung von $q=1$ befasst, soll im Folgenden vorgestellt werden.

3.3 Field-Similarity

Das Field-Similarity Verfahren wird in der Literatur [ZLBL04] und [SLP02] als Ähnlichkeitsmaß für den Zeichenkettenvergleich in Verbindung mit PC-Filter vorgeschlagen. Dieses Verfahren vergleicht zwei Eingangszeichenketten A und B, indem die Zeichenketten in einzelne Zeichen zerlegt werden und dabei die bisherige Auftrittsanzahl eines Zeichens jeweils vermerkt wird. Zum Beispiel wird aus der Zeichenkette „aabcdab“ die Menge $\{a1, a2, b1, c1, d1, b2\}$. Anschließend werden die Tupel A und B geschnitten. Die Schnittmenge enthält nun die Zeichen, die in beiden Zeichenketten in gleicher Auftrittsreihenfolge vorkommen. Nehmen wir wieder das Beispiel:

$A = \text{“Mainz“}$

$B = \text{“Heinz“}$

$T_a = \{M1, a1, i1, n1, z1\}$

$T_b = \{H1, e1, i1, n1, z1\}$

$T_a \cap T_b = \{i1, n1, z1\}$

Hier wird ersichtlich, dass es sich beim Ähnlichkeitsmaß um ein Q-Gram Verfahren mit $q = 1$ handelt. Die Faktorbildung unterscheidet sich allerdings zum vorher beschriebenen Verfahren. Dementsprechend weicht das Ergebnis von dem mit dem vorgestellten Q-Grams-Verfahren ab. Zur Bestimmung des Ähnlichkeitsfaktors wird die Anzahl der im resultierenden Tupel enthaltenen Zeichen in Relation zu der Zeichenanzahl der Originaltupel gesetzt. Es wird der kleinere Wert gewählt, da die Ähnlichkeit der beiden Zeichenketten zueinander nicht größer sein kann als die kleinste Ähnlichkeit des resultierenden Tupels zu einem Originaltupel:

$$\text{sim}(T_a, T_b) = \min\left\{\frac{|T_a \cap T_b|}{|T_a|}, \frac{|T_a \cap T_b|}{|T_b|}\right\}$$

hier: $\text{sim}(T_a, T_b) = \frac{3}{5} \doteq 0.6$ entgegen $\frac{3}{7} \doteq 0.42$ mit Q-Grams $q=1$

Beispiel 3.5: unähnliche Zeichenfolge mit Field-Similarity

Wie bereits erwähnt ignoriert dieses Verfahren die Reihenfolge des Auftretens von Zeichen einer Auftrittszahl. Dies führt dazu, dass auch $B = \text{“Hinze“}$ einen Ähnlichkeitsfaktor von $\frac{3}{5}$ hat.

3.4 Vergleich von Q-Gram zu Field-Similarity

Die folgende Tabelle gibt einen Überblick über die erzeugten Ähnlichkeitsfaktoren von Q-Grams mit unterschiedlichen q Werten und Field-Similarity.

Eingabe 1: (Zeichendreher) $A = \text{“Hanna“}$, $B = \text{“Hanan“}$

Eingabe 2: (Austausch von zwei Zeichen) $A = \text{“Henne“}$, $B = \text{“Hanna“}$

Eingabe 3: (unähnliche Zeichenfolge) $A = \text{“Mainz“}$, $B = \text{“Heinz“}$

Eingabe 4: (stark unähnliche Zeichenfolge) $A = \text{“Mainz“}$, $B = \text{“Hinze“}$

Eingabe 5: (unähnliche Zeichenfolge, gleicher Zeichenvorrat) $A = \text{“Heinz“}$, $B = \text{“Hinze“}$

Eingabe 6: (ausgelassener Buchstabe) $A = \text{“Hana“}$, $B = \text{“Hanna“}$

Eingabe	$q = 3$	$q = 2$	$q = 1$	Field-Sim
1	0,27	0,5	1	1
2	0,07	0,2	0,42	0,6
3	0,27	0,33	0,42	0,6
4	0,07	0,2	0,42	0,6
5	0,07	0,2	1	1
6	0,62	0,83	0,8	0,8

Tabelle 3.1: Vergleich der Ähnlichkeitsfaktoren

Wie wir hier sehen können, liefert Field-Similarity und $q=1$ Vergleichsweise ähnliche Werte. Der Unterschied bei 2 bis 4 ergibt sich aus der Bewertung hinsichtlich der Gesamteingabe bei Q-Grams entgegen der Einzelbetrachtung von Field-Similarity. Beide liefern ein Verhältnis der enthaltenen Zeichen beider Eingabefolgen. Das liefert im Fall 1 eine gute Erkennung des Zeichendreher und des fehlenden Zeichens in Fall 6, führt aber zu einer starken Fehlererkennung im Fall 4 und 5. $q=2$ gibt durchweg zufrieden stellende Ergebnisse, scheitert jedoch an der Erkennung des Zeichendreher, dessen Ähnlichkeitsfaktor zu niedrig ist, um solche Fehler automatisch entdecken zu können. $q=3$ folgt am ehesten der intuitiven Ähnlichkeitsverteilung, scheitert jedoch ebenfalls an dem Zeichendreher. Zudem scheinen sich die Faktoren mit zunehmenden q stark zu reduzieren für kurze Eingabefolgen.

Insgesamt lässt sich feststellen, dass sich Field-Similarity eignet, um Zeichendreher und Zeichenketten mit gleichem Zeichenvorrat zu erkennen. Q-Grams eignet sich zur Bewertung der Folge der Zeichen. Es biete sich an diese Eigenschaften zu kombinieren, um sowohl Tippfehler finden zu können als auch falsche Erkennungen durch ähnliche Zeichenvorräte zu vermeiden. Des Weiteren liegt es in der Definition der Verfahren, eine Menge an Wörtern, zum Beispiel Sätze, als ein einziges Wort zu behandeln. Durch den Einsatz des Atomic-String-Prinzips können die Verfahren auch auf Sätze angewandt werden.

3.5 PC-Filter

Das gewählte Verfahren zur Duplikatserkennung entstammt dem Artikel [ZLBL04]. Es handelt sich um ein Verfahren, das in drei Hauptschritten eine Menge an Tupeln auf Duplikate untersucht. Das hier vorgestellte Verfahren bedarf einer Menge an Tupeln, einer Vorgabe für die Anzahl an Partitionen, die Größe des einzubeziehenden Partitionsrandbereiches und dem Genauigkeitsgrad als Schranke für den Ähnlichkeitsvergleich. Optional ist das Angeben verschiedener Attributgewichte (s.u.) und Ähnlichkeitsmaße für jedes Attribut.

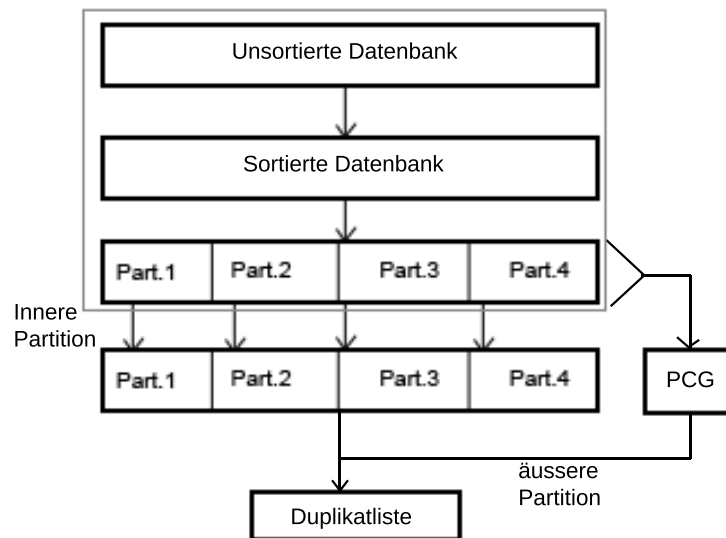


Abbildung 3.1: PC-Filter Übersicht

Die Schritte sind:

- Sortieren und Partitionieren
- Erstellen eines Partitionsgraphen
- Partitionen vergleichen

Das Verfahren benötigt *mindestens ein* Ähnlichkeitsmaß, dessen Ähnlichkeitsfaktor im Intervall $[0,1]$ liegen sollten, um die hier dargestellten Formeln anwenden zu können. Werden Tupel mit mehr als einem Attribut verglichen, kann es notwendig sein, Attribute zu priorisieren. Diese fließen stärker in die Ähnlichkeitsfaktorbildung zwischen zwei Tupeln ein. Diese Priorisierung wird mit dem **Attributgewicht** W vorgenommen. Für jedes Attribut wird ein solches Gewicht festgelegt. Die Summe aller Gewichte ist 1.

$$1 = \sum_{i=1}^n (W_i)$$

Es sei hier die Definition von $\text{sim}(T_a, T_b)$ angenommen wie unter Field-Similarity (vgl. 3.3). Der Ähnlichkeitsfaktor zweier Tupel A und B berechnet sich dabei aus der Summe aller Produkte aus Attributgewicht W_i und Ähnlichkeitsfaktoren der Attribute mit i als Zähler für das aktuelle Attribut:

$$\text{Sim}(A, B) = \sum_{i=1}^n (W_i * \text{sim}_i(T_{a,i}, T_{b,i}))$$

Um die Datensätze effektiv zu vergleichen, wird eine Art Transitivität definiert. Ziel ist aus der Ähnlichkeit von Tupel A zu B sowie B zu C die Ähnlichkeit von A zu C abzuschätzen. Gehen wir von diesem Ziel aus, so lässt sich aus den Ähnlichkeitsfaktoren ein Dreieck konstruieren.

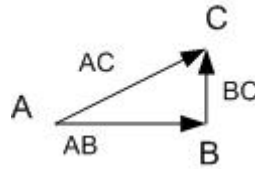


Abbildung 3.2: Ähnlichkeitsfaktoren am Dreieck

An diesem Dreieck können wir die allgemeine Dreiecksungleichung anwenden:

$$|a - b| \leq c \leq a + b$$

nach Abbildung 3.2:

$$a = AB$$

$$b = BC$$

$$c = AC$$

Auflistung 3.1: Allgemeine Dreiecksungleichung

Die Begrenzungen, die diese Ungleichung vorgibt, bezeichnet PC-Filter als *Upper Bound* (UB) und *Lower Bound* (LB). Dabei macht PC-Filter davon Gebrauch, dass zwei Dreiecksseiten durch die Intervallbeschränkung maximal die Summe 2 haben können, und der Betrag der Subtraktion maximal 1 ist. Setzt man nun die Ähnlichkeitsfaktoren unter Beachtung der Intervallgrenzen ein, ergibt sich die Ungleichung:

$$(AB + BC) - 1 \leq AC \leq 1 - |AB - BC|$$

$$AB = \text{Sim}(A, B)$$

$$BC = \text{Sim}(B, C)$$

$$AC = \text{Sim}(A, C)$$

Auflistung 3.2: Dreiecksungleichung nach PC-Filter

Die Umkehrung der Vergleichsoperationen stellt sich dadurch ein, dass durch die hier verwendete Schreibweise, es zu einer komplementären Darstellung der Terme kommt. Der Grund liegt darin, dass $(AB + BC)$ zwischen 0 und 2 schwankt. Durch Abzug von 1 wird ein ähnliches Verhalten wie vom ursprünglichen Betragsterm erzielt, dessen maximaler Wert 1 nicht überschreitet. Die gewählten Terme verhalten sich wie die allgemeine Dreiecksungleichung, sind aber weniger streng. Wir betrachten folgendes Beispiel:

$$AB = 0,2$$

$$BC = 0,3$$

$$-0,5 \leq AC \leq 0,9$$

entgegen der allgemeinen Form: $0,1 \leq AC \leq 0,5$

Beispiel 3.6: Beispiel für Dreiecksungleichung nach PC-Filter

Es stellt sich für unser AC ein größerer Bereich ein. Damit wird mit der Ungleichung nach PC-Filter berücksichtigt, dass je unterschiedlicher die Tupel untereinander sind, desto weniger über die Ähnlichkeit von A zu C ausgesagt werden kann.

PC-Filter beschreibt auf Grund der Ungleichung aus Auflistung 3.2 die Lower und Upper Bound wie folgt:

$$LB_B(A, C) = Sim(A, B) + Sim(B, C) - 1$$

$$UB_B(A, C) = 1 - |Sim(A, B) - Sim(B, C)|$$

Lemma 1 aus [ZLBL04] besagt: Erfüllen die generierten Ähnlichkeitsfaktoren die Dreiecksungleichung, insbesondere auch die Ungleichung aus Auflistung 3.2, so gelten die LB und UB Gleichungen.

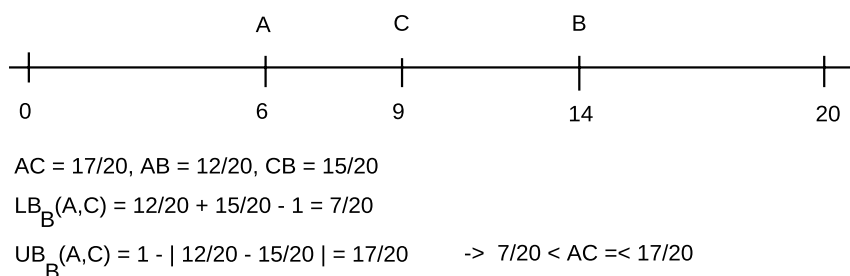


Abbildung 3.3: Lower und Upper Bound am Zahlenstrahl

Ein solcher Ähnlichkeitsfaktor, der die Ungleichung aus Auflistung 3.2 sicher erfüllt, muss folgende Punkte mindestens erfüllen:

- durch geschlossenes Intervall begrenzt
- gleiches Verfahren zur Faktorbestimmung für jedes Tupelpaar

Die Intervallbegrenzung ist notwendig, um die höchste und niedrigste Ähnlichkeit festlegen zu können. Außerdem muss das Verfahren zur Bildung des Ähnlichkeitsfaktors, das umfasst auch das Ähnlichkeitsmaß, für jedes Tupelpaar gleich sein. Wie bereits in Abschnitt 3.4 zu sehen war, bilden selbst ähnliche Verfahren andere Faktoren. Wird für verschiedene Tupelpaare unterschiedliche Verfahren verwendet, sind diese nicht vergleichbar. Es könnte kein Dreieck gebildet werden, da zum Beispiel AC keinen Zusammenhang zu BC hätte.

Q-Grams und Field-Similarity erfüllen sowohl die allgemeine Dreiecksungleichung als auch die Ungleichung nach Auflistung 3.1.

Aus der UB und dem LB lassen sich die Eigenschaften (1) und (2) konstruieren. Dabei sei G die vorgegebene Genauigkeit:

- (1) Wenn $LB_B(A, C) \geq G$, dann ist A und C ein Duplikat (kleinster Abstand überschreitet Grenze)
- (2) Wenn $UB_B(A, C) < G$, dann sind A und C **kein** Duplikat (größter Abstand unterhalb der Grenze)

Partitionen ordnen wir ein in innere und äußere Partitionen. Dabei sind alle die Partitionen, zu denen ein Tupel R nicht gehört, Teil der äußeren Partitionen von R .

Vergleichen wir nun ein beliebiges Tupel R mit einer Partition, gelten die Eigenschaften (3) und (4). Dabei definieren wir je ein Tupel einer Partition als den so genannten *Anchor Record* (AR). Es bietet sich an, das mediane Tupel einer Partition zu wählen. *MinSim* sei dabei definiert als die kleinste Ähnlichkeit zwischen dem AR und allen Datensätzen der Partition, zu dem AR gehört.

- (3) Wenn $Sim(R, AR) + MinSim - 1 \geq G$, dann ist R definitiv Duplikat zu **allen** Datensätzen der Partition (folgt aus Ungleichung, vergleiche Definition von LB)
- (4) Wenn für jedes Tupel D der Partition gilt $(1 - \min\{|Sim(R, AR) - Sim(D, AR)|\}) < G$, dann ist R kein Duplikat zu einem Datensatz aus der Partition (vergleiche Definition von UB)

Mit diesen Vorbemerkungen können wir die einzelnen Schritte ausführen.

Schritt 1: Sortieren. Die Datenbank, bzw. im folgenden zur Einfachheit auf Tabelle reduziert, wird nach einem oder mehreren Feldern geordnet, nach denen die Wahrscheinlichkeit hoch ist, dass duplizierte Tupel nah beieinander liegen, zum Beispiel nach Nachname oder Orten. Laufende Zahlen eignen sich eher nicht. Anschließend wird die sortierte Tabelle in eine Menge von k Partitionen aufgeteilt, wobei k vorgegeben ist. Von der Größe k hängt unter anderem ab, wie viele Iterationen das Erstellen des PC-Graphen benötigt. Während der Aufteilung ist ebenfalls die Bestimmung der *Anchor Records* vorzunehmen und eine Ähnlichkeitsliste zu erstellen, in der pro Partition die Ähnlichkeit vom AR zu jedem Datensatz abgelegt wird. Dies erleichtert das Anwenden der Eigenschaften (3) und (4) in den folgenden Schritten. Um die AR zu bestimmen, empfiehlt es sich, das mediane Tupel einer Partition zu wählen, da bei guter Wahl der Sortierung im optimalen Fall eine Gleichverteilung der Ähnlichkeiten zum AR innerhalb der Partition vorliegt

Schritt 2: PC-Graph erstellen. Der *Partition-Comparison-Graph* (PCG) dient dazu, Partitionen zu bestimmen, die untereinander geprüft werden müssen, da deren Randelemente andeuten, dass durch das Aufteilen in Schritt 1, Tupel in äußeren Partitionen eventuelle Duplikate haben. Dazu dient die Vorgabe des Partitionsrandbereiches N . Vom Anfang und vom Ende der Partition werden je $\frac{N}{2}$ Tupel gewählt, die so genannten *Delimiting Records* (DR). Jeder DR führt eine Liste an Partitionen. Aus dieser Liste werden alle Partitionen entfernt, die die Eigenschaft (4) zu DR nicht erfüllen.

$$PartListeDR = I - \{P_{DR}\} - \{P \mid P \text{ äußere Partition von DR die Eigenschaft (4) nicht erfüllt}\}$$

I ist dabei die Menge aller Partitionen, und P_{DR} bezeichnet die eigene innere Partition von DR. Aus diesen Partitionslisten kann nun die *Partition-Comparison-Range* (PCR) erstellt werden. Die untere Grenze setzt sich zusammen aus dem Minimum aller nicht

leeren Listen der ersten $\frac{N}{2}$ DR zu einer Partition, die obere Grenze analog aus dem Maximum der letzten $\frac{N}{2}$ DR zu einer Partition.

$$PCR_{\text{untereGrenze}} = \min\{\min\{PartList(DR_1)\}, \dots, \min\{PartList(DR_{\frac{N}{2}})\}\}$$

$$PCR_{\text{obereGrenze}} = \max\{\max\{PartList(DR_{\frac{N}{2}+1})\}, \dots, \max\{PartList(DR_N)\}\}$$

Sind alle Listen der DR leer, so ist dies als [,\$,\$] zu vermerken. Es bildet sich eine Tabelle wie in diesem Beispiel:

Partitionen: 1, 2, 3, 4

Partition	$\cup$ PartList
1	2, 4
2	4
3	<leer>
4	1, 2

Tabelle 3.2: Vereinigung der PartListen pro Partition

Partition	PCR _{untereGrenze}	PCR _{obereGrenze}
1	2	4
2	4	4
3	\$	\$
4	1	3

Tabelle 3.3: PCR Beispiel

Zum Erstellen des PCG wird jede Partition als Knoten erzeugt. Von jedem Knoten geht nun eine Linie zu den Knoten, dessen Partition in der PCR liegen. Dabei ist darauf zu achten, dass es sich beim PCG um einen ungerichteten Graphen handelt.

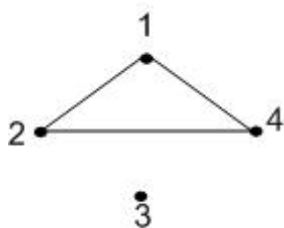


Abbildung 3.4: PCG Beispiel

Schritt 3: Partitionen vergleichen. Das Vergleichen der Partitionen erfolgt in zwei Schritten:

Innerhalb der Partition: Zunächst wird innerhalb der Partitionen mit Hilfe der Eigenschaften (1) und (2) jeder Datensatz verglichen. Dazu kann die in Schritt 1 erzeugte Ähnlichkeitsliste verwendet werden, indem in den Eigenschaften für das Tupel B der AR eingesetzt wird. Damit kann dieser Vergleich durch einen Sweep, also dem alleinigen

Durchgehen der Ähnlichkeitsliste, erfolgen. Kommen (1) und (2) zu keinem Ergebnis, müssen die Datensätze direkt verglichen werden (ohne Transitivitätsausnutzung oder mit genauerem Ähnlichkeitsmaß).

Außerhalb der Partition: Wurden alle Partitionen durchsucht, wird der PCG abgearbeitet. Für jeden Knoten, also Partition, werden die verbundenen Partitionen geprüft. Dabei wird für jeden Datensatz R der inneren Partition mit Eigenschaft (3) die Ähnlichkeit zu jedem AR der verbundenen äußeren Partitionen geprüft. Kommt dies zu keinem Schluss, wird mit Eigenschaft (4) die Ähnlichkeiten der ganzen benachbarten Partitionen zu R verglichen. Kommen (3) und (4) zu keinem Ergebnis, ist R mit Eigenschaft (1) und (2) und jedem Eintrag der benachbarten Partition zu vergleichen. Nachdem alle verbundenen äußeren Partitionen geprüft wurden, kann der Knoten der inneren Partition und die damit verbundenen Linien gelöscht werden.

Wurden alle PCG Verbindungen abgearbeitet, ist die Suche abgeschlossen. Wurde durch die Anwendung von Eigenschaften (1) und (3) oder durch direkten Vergleich festgestellt, dass das Tupel R ein Duplikat zu einem oder mehreren weiteren Tupeln ist, so sollen diese Tupelpaare in der Duplikatsliste abgelegt werden. Diese Ergebnisliste ist gruppiert und kann dann zur Fehlerkorrektur verwendet werden.

Kapitel 4

Konzeption

In diesem Kapitel wird ein Konzept für eine Werkzeugsuite zur Datenbereinigung einer Oracle Datenbank vorgestellt.

4.1 Überblick

In den vorherigen Kapiteln haben wir Verfahren vorgestellt, die spezielle Teilbereiche der Datenbereinigung abdecken. In diesem Kapitel sollen nun Werkzeuge entwickelt werden, die den Benutzer bei der Datenbereinigung unterstützen. Dazu definieren wir zunächst das Werkzeug als ein Instrument, das Verfahren einsetzt, um ein höheres Ziel zu erreichen. So setzt zum Beispiel ein Werkzeug zur Duplikatsbereinigung die Verfahren zur Duplikatserkennung, Verfahren zur Erzeugung von Ähnlichkeitsfaktoren und das Mergingverfahren ein, mit dem Ziel, Duplikate aus der Datenbank zu entfernen.

Damit der Benutzer solche Werkzeuge verwenden kann, bedarf es einer Schnittstelle. Diese stellt das Anwendungsprogramm dar. Dieses Programm verwaltet die nötigen Informationen, die die Werkzeuge benötigen, und lässt den Benutzer mit den Werkzeugen interagieren. Das Anwendungsprogramm ist damit ein Framework für Werkzeuge. Durch die daraus resultierende Zentralisierung der Werkzeuge ergibt sich die Werkzeugsuite, die im Folgenden vorgestellt wird.

Ein wiederkehrendes Detail wird die Unterbrechbarkeit der Werkzeuge sein, die nach 1.2 gefordert wird. Dieses Thema soll im Abschnitt 4.8 ausgeführt werden. Im Zusammenhang stehende Funktionalitäten erscheinen aber bereits in den zugehörigen Abschnitten.

4.2 Architektur

Konzipiert werden soll eine Werkzeugsuite zur Bereinigung einer Oracle Datenbank als Benutzeranwendung. Dies setzt sich zusammen aus dem benutzerinteraktiven Teil, einer implizierten Steuereinheit und aus den Werkzeugen, die auf den Datenbestand wirken. Zur detaillierteren Diskussion teilen wir diese Zusammensetzung auf in:

- Die Benutzerebene
- Die Verarbeitungsebene

Dabei umfasst die Benutzerebene die Interaktionsschnittstelle mit dem Benutzer, was auch die Wahl der Werkzeuge beinhaltet. Die Verarbeitungsebene stellt den operativen Part der Werkzeuge auf den Datenbestand dar. Anders ausgedrückt entfällt auf die Benutzerebene die graphische Darstellung sowie eine Sammlung an Informationen der verfügbaren Werkzeuge („Informationen“ umfasst hierbei zum Beispiel Konfigurationsmöglichkeiten und Parameter). Die Verarbeitungsebene wiederum beinhaltet den Datenbestand und die darauf operierenden Prozeduren der Werkzeuge. Es zeigt sich, dass der Werkzeugpart Teilmenge beider Ebenen ist.

Der Rahmen der Benutzerebene stellt der Client dar. Dieser Client stellt dem Benutzer Interaktionsmöglichkeiten zu Verfügung, um Werkzeuge auf eine Datenbank anzuwenden. Teil des Client ist folglich die Bedienoberfläche. Zusätzlich zu den allgemeinen Bedienoptionen des Frameworks sind die Eigenschaften der Werkzeuge darzustellen. Ebenso beschreibt dieses Segment, wie die Prozeduren der Werkzeuge auszuführen sind. Es ergibt sich also eine Abhängigkeit mit der Verarbeitungsebene. Abbildung 4.1 zeigt den Aufbau dieser Ebenen.

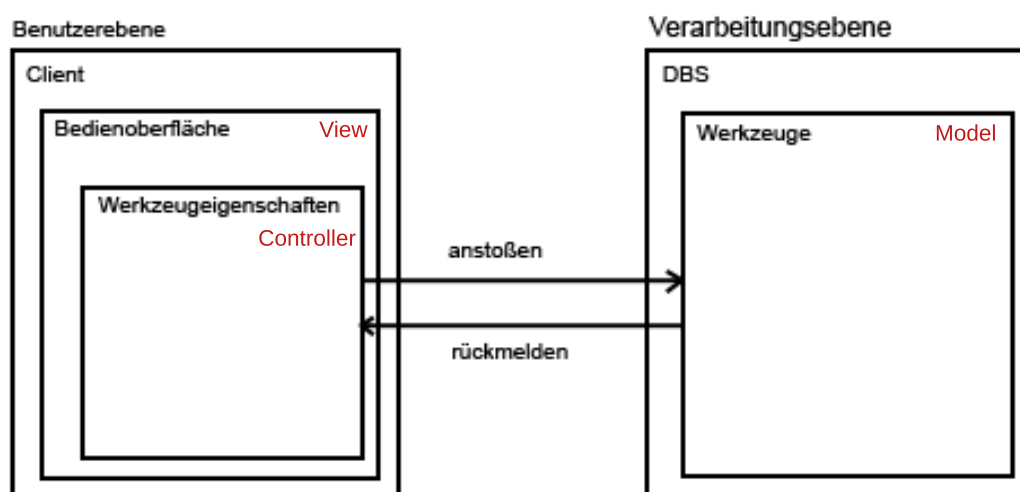


Abbildung 4.1: Architekturebenen

In der dargestellten Verarbeitungsebene bildet das Datenbanksystem (DBS) den Rahmen. Dies hat zwei Gründe:

Zum Einen haben wir den Datenbestand als Teil der Verarbeitungsebene definiert. Die Werkzeuge operieren dabei auf einer Teilmenge dieses Datenbestandes. Der Datenbestand selber ist nur eine Teilmenge des DBS.

Zum Anderen erlaubt uns das Oracle DBS die Werkzeugprozeduren direkt auf der Datenbank zu Programmieren, zum Beispiel unter Ausnutzung von *PL/SQL*. Generell geschehen alle Zugriffe auf den Datenbestand letztendlich nur über das DBS. Damit können wir eine „Anstoßen“ und „Rückmelden“ Beziehung zwischen den beiden Ebenen annehmen. Dabei besagt das „Anstoßen“, dass eine Anforderung an das Werkzeug erfolgt, eine Anfrage an die Datenbank zu stellen. Dies kann schon ein einfacher SQL Befehl sein. Die Rückmeldung aus dem Datenbestand wird anschließend durch das Werkzeug behandelt und der Benutzerebene zurückgemeldet. Entfällt hier der Werkzeugpart auf der Datenbank, so wird dennoch die Rückmeldung des SQL Befehls weitergeleitet.

Für die Teilblöcke der Benutzer- und Verarbeitungsebene kann das, aus der Softwaretechnik bekannte, Model-View-Controller-Muster angewendet werden. In Abbildung 4.1 ist eine mögliche Zuordnung hervorgehoben.

Das „Anstoß“-Prinzip erfordert eine Ablaufsteuerung, die die Kontrolle über die Werkzeuge übernimmt. Zudem bedarf es Vorgaben, nach denen die Anwendung reagiert.

Diesem Modell nach entfallen solche Einstellungen und die **Ablaufsteuerung** auf die Benutzerebene. Die anfallenden Einstellungen beinhalten dabei die globalen Anwendungseinstellung als auch die spezifischen Werkzeugeinstellungen. Die Gesamtheit dieser Einstellungen bezeichnen wir als **Konfiguration**. Da Werkzeuge für den Benutzer nur verfügbar sind, wenn diese sowohl der Verarbeitungsebene als auch der Benutzerebene bekannt sind, können wir eine Rahmenstruktur für die Einbindung von Werkzeugen zur Verfügung stellen. Dieser **Framework** organisiert und verwaltet die Konfiguration und sorgt für die Ausführung der Werkzeuge gemäß der Ablaufsteuerung.

4.2.1 Anwendung des MVC-Musters

Das „View“ Element des MVC-Musters ist für die graphische Darstellung der Interaktionsschnittstellen zuständig. Dessen Interaktionen werden vom „Controller“ ausgewertet und die Operationen des „Model“ angestoßen, die auf Datenquellen zugreifen. Dies entspricht bereits größtenteils der zuvor beschriebenen Kommunikation der Ebenen. Wir definieren das „View“ Element als die Darstellung der Bedienoberfläche. Die Vorkonfiguration der Anzeige übernimmt für uns der „Controller“-Part. Dieser setzt sich zusammen aus dem frameworksteuernden Segment und dem werkzeugsteuernden Teil. Das Frameworksegment folgt dem vorgegebenen Programmablauf und ist für die Rahmenfunktion der Anwendung notwendig. Das Werkzeugsegment setzt sich zusammen aus allen im Framework registrierten Werkzeugen. Dieses umfasst wiederum die zur Anzeige der Werkzeugeigenschaften notwendigen Informationen als auch die werkzeugsteuernden Operationen. Das „Model“ schließlich ist für den Zugriff auf den Datenbestand zuständig. Die vom „Controller“ aufgerufenen Operationen werden hier abgearbeitet und eine Rückmeldung zurückgeliefert.

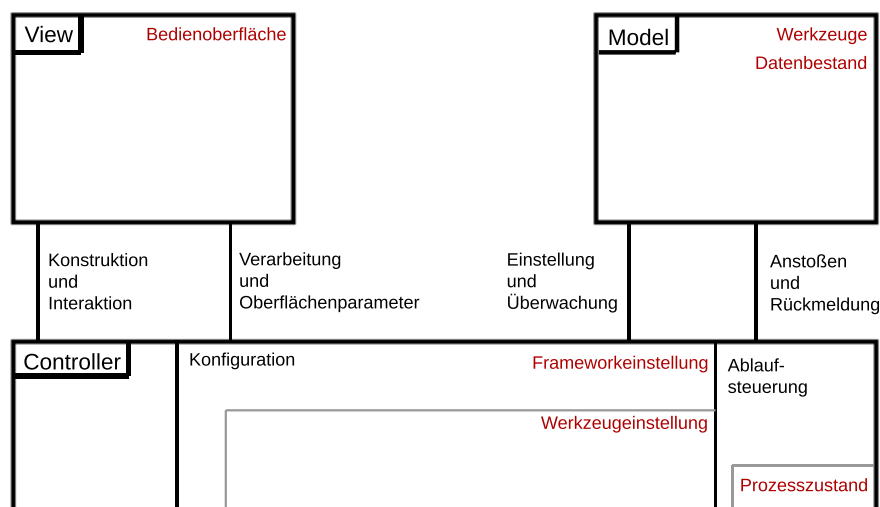


Abbildung 4.2: MVC Abhängigkeitsbeispiel

4.2.2 Blockstruktur

Neben der Aufteilung in Ebenen lassen sich die Funktionseinheiten der Anwendung in Funktionsblöcke aufspalten. Durch eine solche Blockstruktur erreichen wir zum Einen eine übersichtliche Aufteilung der Teilsysteme, und zum anderen können wir die Verbindung der Blöcke zueinander als Final-State-Machine Interpretieren. Dies erlaubt uns jederzeit in unserem Programmablauf den aktuellen Zustand festzuhalten. Jeder Funktionsblock kennt seinen Zustand, in dem er sich befindet. Auf der übergeordneten Ebene können wir beim Wechsel eines Funktionsblockes einen **Zustandswechsel** verzeichnen. Für die übergeordnete Einheit bedeutet ein Zustandswechsel folglich, dass die Zustandsinformationen des zuletzt aktiven Blockes nicht mehr relevant sind. So sind beispielsweise die Oberflächenoptionen der „Konfiguration“ uninteressant, wenn sich die Anwendung in der „Ablaufsteuerung“ befindet.

Aus der Beschreibung des Frameworks sind uns für die Benutzerebene schon die beiden Blöcke „Konfiguration“ und „Ablaufsteuerung“ bekannt. Dieses soll im Abschnitt 4.3 weiter ausgeführt werden. Zunächst betrachten wir die Blockstruktur für ein Werkzeug.

Das Werkzeug setzt sich zusammen aus den Blöcken „Konfiguration“ und „Ausführung“ auf der Benutzerebene und den Blöcken „Fehlererkennung“ und „Fehlerkorrektur“ auf der Verarbeitungsebene.

Der Konfigurationsblock gibt dem Benutzer die Möglichkeit, das Werkzeug auf seine Bedürfnisse anzupassen oder wichtige Parameter voreinzustellen. Der Ausführungsblock überwacht die Operationen des Verarbeitungsblockes. Das heißt, der Ausführungsblock ist für das Abarbeiten der beiden Blöcke der Verarbeitungsebene zuständig und wertet deren Rückmeldungen aus. Dazu zählt auch die Bereitstellung der Benutzerinteraktion. Jeder Ausführblock der Werkzeuge ist somit auch Teil der Ablaufsteuerung. Näheres zum Ausführungsblock folgt im Abschnitt 4.3.

Der Fehlererkennungsblock beinhaltet Operationen zur Erkennung von Fehlern auf dem Datenbestand und meldet diese dem Ausführungsblock zurück. Der Fehlerkorrekturblock wertet das Ergebnis der Erkennung aus, führt die Korrekturmaßnahmen durch und schickt dem Ausführungsblock eine Rückmeldung.

Über allen steht die Konfiguration, in denen die Einstellungen der Werkzeuge aus dem Konfigurationsblock abgelegt werden und diese den restlichen Blöcken zur Verfügung stellt.

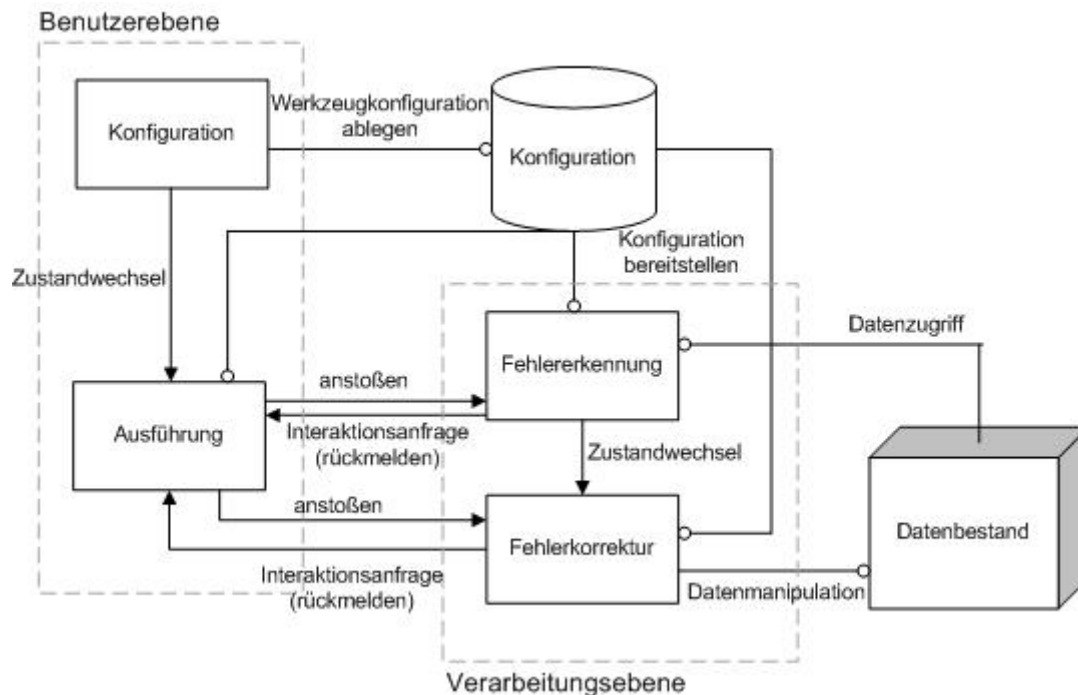


Abbildung 4.3: Funktionsblöcke von Werkzeugen

4.2.3 Datenspeicher

Dem MVC zur Folge hat nur der „*Model*“-Teil Zugriff auf externe Datenquellen. Hier soll gezeigt werden, dass diese Definition für unsere Architektur ausreichend ist.

Der „*Model*“-Teil befindet sich hier auf der Verarbeitungsebene und hat dort Zugriff auf den Datenbestand. Da sich alle andere Elemente auf Benutzerebene befinden, ist die erste Vorgabe bereits erfüllt. Unter Beachtung der unter Abschnitt 1.2 geforderten Konfigurationseigenschaften und einer eventuellen Unterbrechungs-Option, wie unter 4.8 beschrieben, ist es notwendig, auch auf Benutzerebene einen Datenspeicher zu führen. Dieser kann dabei aber so gestaltet werden, dass es sich hierbei nur um eine Sammlung von Zuständen des „*Controller*“ handelt. Dieser kann, wenn auch extern gelagert, direkt wieder in den „*Controller*“ eingelesen werden. Daher kann diese Funktionalität als Erweiterung der „*Controller*“-Funktionalität aufgefasst werden.

Es bleibt zu klären, für welche Daten des „*Controller*“ eine persistente Speicherung Sinn macht. Das wäre zum Einen die **Konfiguration**. In der Konfiguration sind alle vom Benutzer vorgegebenen Einstellungen enthalten. Wird diese gespeichert, so braucht diese nicht erneut manuell Eingegeben werden, wenn die Anwendung zurückgesetzt wird, sondern kann wieder geladen werden. Zum Anderen macht das Ablegen des Zustandes der Anwendung Sinn, da hiermit der zuletzt aktive Block gemäß 4.2.2 fortgesetzt werden kann. Es reicht, den Zustand der „Ablaufsteuerung“ zu speichern, da sich die Zustände der „Konfiguration“ mit dem Erzeugen der zuvor erwähnten Konfiguration beschäftigen und mit dem Speichern eben solcher nicht mehr relevant sind. Der Zustand der „Ablaufsteuerung“ ist der **Prozesszustand**.

Wie aus 4.2.2 bereits zu entnehmen und wie es sich im Folgenden zeigen wird, besteht die Konfiguration aus einzelnen Blöcken. Jedes Werkzeug hat seinen eigenen Konfigurationsblock. Dazu kommt der Block für allgemeine Einstellungen (im Folgenden auch

Frameworkeinstellungen). Es bietet sich an, eine lokale Speicherstruktur zu nutzen, mit derer es möglich ist, diese Blöcke geordnet zu speichern. Idealer Weise lassen sich diese Daten in einer Datenbank ablegen, da diese bereits geordnete Strukturen bereitstellt. Eine Portabilität wäre mit dieser Lösung allerdings sehr gering. Als besseres Beispiel sei die „Extensible Markup Language“ *XML* erwähnt, mit derer es möglich ist, die Daten wie in einer lokalen Datenbank abzulegen und zu verwalten. Zudem können diese Dokumente mit sehr geringem Aufwand zwischen verschiedenen Clients ausgetauscht werden.

4.3 Ablaufbeschreibung

Der bisherige Aufbau des Frameworks besteht aus den Blöcken „Konfiguration“ und „Ablaufsteuerung“. Aus diesen Blöcken, die den Aufbau beschreiben, sind nun Funktionsblöcke abzuleiten, die einen Ablauf beschreiben. Aus den Architekturebenen der Abbildung 4.1 ist eine Trennung zwischen dem Client, also der dem Framework zugrunde liegenden Instanz, und dem DBS zu erkennen. Um diese Verbindung herzustellen, bedarf es dem Block „Initialisierung“. Dieser Block muss folglich vor der ersten Kommunikation mit der Verarbeitungsebene abgelaufen sein. Das heißt, er kann vor oder nach dem Konfigurationsblock ablaufen. Aus Gründen der Übersichtlichkeit soll zunächst angenommen werden, dass er vor dem Konfigurationsblock läuft und damit bereits eine Verbindung zur Verarbeitungsebene besteht.

Der Block „Ablaufsteuerung“ umfasst die Ausführung der Werkzeuge, deren Zustandsüberwachung, die eigene Zustandsüberwachung und den Ablaufplan. Während sich die Anwendung in diesem Block befindet, werden also Werkzeuge ausgeführt. Dabei regelt der Ablaufplan, welche Werkzeuge ausgeführt werden. Da die Ausführung der Werkzeuge Teil des Ausführungsblocks des jeweiligen Werkzeuges ist, behandeln wir diesen separat (siehe dazu auch 4.2.2 und 4.4). Der Ablaufplan enthält neben den statischen Programmablauf auch die Aufrufliste für die Werkzeuge. Diese Liste bezeichnen wir als „**Prozessliste**“. Diese wurde während der „Konfiguration“ erstellt und wird später behandelt. Der Zustand des Ablaufs wird im „**Prozesszustand**“ festgehalten. Darunter fällt die Position in der Prozessliste, der Zustand des Ausführungsblocks des jeweiligen Werkzeuges und der eigene Zustand.

Die Elemente Ausführung und Zustandsüberwachung der „Ablaufsteuerung“ bilden den „Prozess“ Block, in dem die Anwendung selbsttätig agiert und Zustandsänderungen überwacht.

Der Ablauf der Anwendung entspricht damit den Blöcken „Initialisierung“, „Konfiguration“ und „Prozess“.

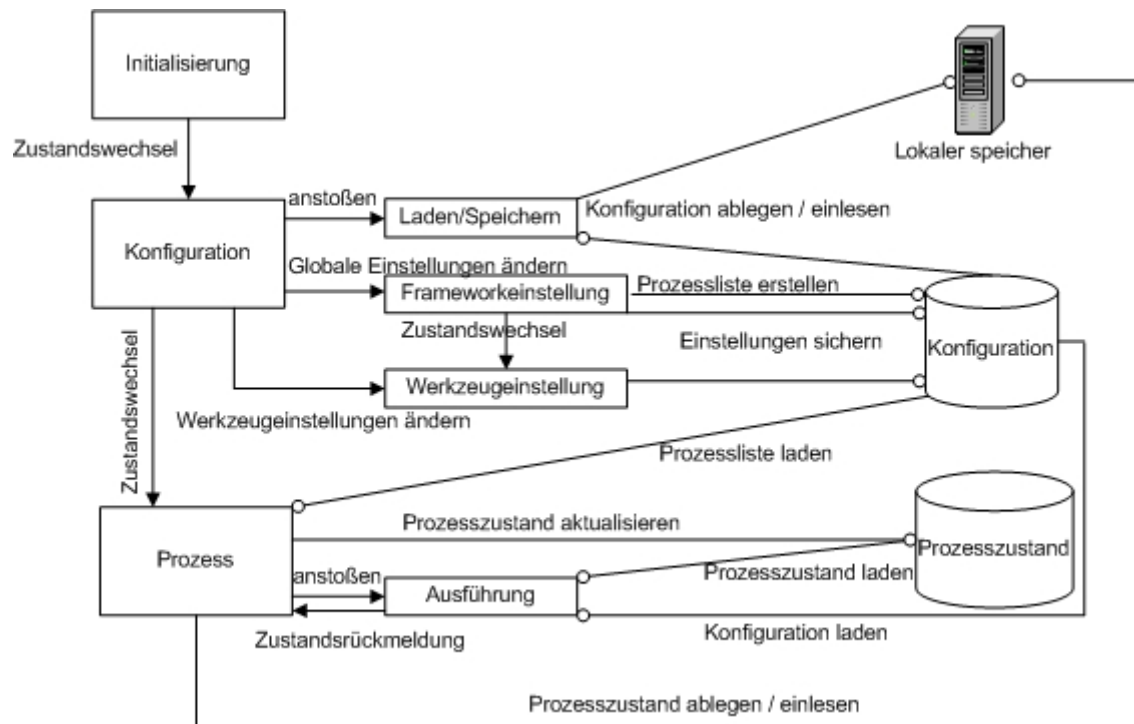


Abbildung 4.4: Anwendung Funktionsblöcke

Im Initialisierungsblock wird die notwendige Datenverbindung hergestellt. Darunter fällt die Authentifizierung an der Datenbank und eventuelle Tests der Verbindung.

Der Konfigurationsblock steht für die Einstellung des Frameworks und der Werkzeuge. Dabei gelten die Frameworkeinstellungen für alle Werkzeuge. Eine Frameworkeinstellung kann zum Beispiel die gewählte Tabelle sein. Eine Werkzeugeinstellung ist beispielsweise die Erkennungsgenauigkeit. Die abgelegte Konfiguration umfasst beide Einstellungen wie in 4.2 definiert. Teil des Konfigurationsblocks ist auch die Erstellung der Prozessliste. Die Prozessliste gibt vor, welche Werkzeuge der Reihenfolge nach ausgeführt werden sollen. Diese muss vor dem festlegen der Werkzeugeinstellungen gefüllt werden, da die Zuordnung der Einstellung zum eingereichten Werkzeug passen muss.

Vor dem Wechsel in den Prozessblock sollen dem Benutzer folgende Aktionen verfügbar sein:

- Laden oder Speichern der Konfigurationsdaten
- Laden oder Speichern des Prozesszustandes
- Aktuelle Konfiguration bearbeiten
- Konfiguration erstellen
- Aktuelle Konfiguration ausführen
- Eventuell aktuellen Prozesszustand fortsetzen

Das Laden oder Speichern der Konfiguration erlaubt es dem Benutzer, die erstellte Konfiguration für späteren Gebrauch abzulegen. Ebenso kann er die geladene Konfiguration

bearbeiten. Ein Sonderfall stellt das Fortsetzen einer Konfiguration dar. Zum Fortsetzen muss eine Konfiguration geladen sein und ein abgelegter Prozesszustand in den Prozessblock geladen werden. Die geladene Konfiguration wird an dieser Stelle fortgesetzt. Siehe dazu auch den Abschnitt 4.2.3 und 4.8. Das neue Ausführen der aktuellen Konfiguration setzt den Prozesszustand zurück.

Das Erstellen einer neuen Konfiguration erlaubt es dem Benutzer, die zu bearbeitenden Tabellen zu wählen sowie den Attributen Eigenschaften zuzuordnen. Dies stellt die Frameworkeinstellung dar. Anschließend wird eine Prozessliste erstellt. Der Benutzer trägt alle Werkzeuge ein, die nacheinander auf die gewählten Tabellen angewendet werden. Nach erstellen der Prozessliste wird für jedes eingetragene Werkzeug deren Eigenschaften geöffnet. Dort können werkzeugspezifische Einstellungen festgelegt werden. Dies ergibt die Werkzeugeinstellung. Der Prozesszustand umfasst die Position der Abarbeitung der Werkzeuge in der Prozessliste im Prozessblock und wird demnach beim Erstellen einer neuen Konfiguration zurückgesetzt.

Nach dem Wechsel zum Prozessblock wird die in der Konfiguration abgelegte Prozessliste abgearbeitet. Dabei werden der Reihe nach die eingetragenen Werkzeuge gemäß der Konfiguration voreingestellt und ausgeführt. Der aktuelle Abarbeitungszustand wird im Prozesszustand festgehalten und kann im Falle einer Unterbrechung zusammen mit der Konfiguration gespeichert werden. Hierzu sei erwähnt, dass Konfiguration und Prozesszustand separat gespeichert werden können. Es ist jedoch nicht empfehlenswert, eine unpassende Konfiguration zu einem Prozesszustand zu laden.

Der Prozessblock startet den Ausführungsblock des aktiven Werkzeuges. Während der Abarbeitung eines Werkzeuges befindet man sich folglich im Werkzeug. Gemäß den Vorgaben aus Abschnitt 1.2 ist eine Benutzerinteraktion ein Sonderfall. Wir betrachten dies also als eine „Exception“ im Werkzeug. Das heißt: Beantragt das Werkzeug eine Benutzerinteraktion, befinden wir uns immer noch im Ausführungsblock des Werkzeugs, und der weitere Prozessablauf wird angehalten. Dies sei anhand folgenden Beispiels erläutert:

Ein Werkzeug zur Attributwertbereinigung ist vor einer Duplikatbereinigung angesetzt in der Prozessliste. Es tritt der Fall ein, dass ein Attributwert nicht genau zugeordnet werden kann. An dieser Stelle hält die Ausführung und fragt den Benutzer. Würde die Ausführung mit der Interaktion warten bis die Prozessliste abgeschlossen ist, würde bei der anstehenden Duplikatbereinigung das Tupel, bei dem zuvor die Ausnahme eingetreten ist, eventuell nicht als Duplikat erkannt werden. Dadurch müsste die Erkennung ein zweites mal erfolgen.

Beispiel 4.1: Benutzerinteraktion im Ausführungsblock

Die Art der Benutzerinteraktion, die ein Werkzeug verlangt, ist dabei von dessen Eigenschaften abhängig. So kann zum Beispiel die Vergabe von Referenzwerten bereits in den Werkzeugeinstellung erfolgen oder aber bei der Ausnahme vonnöten sein. So wäre bei einem Duplikatwerkzeug zum Beispiel zwischen dem Fehlererkennungs- und dem Fehlerkorrekturblocks die interaktive Korrektur von Clusterzusammensetzungen (Clusterkorrektur) nötig.

4.4 Werkzeuge

Es wurde bisher weitestgehend über das den Werkzeugen zugrunde liegende Framework gesprochen. Dieses stellt dem Benutzer Konfigurationsmöglichkeiten zur Verfügung und verwaltet die Werkzeuge. Hier sollen nun die zur Verfügung stehenden Werkzeuge besprochen werden.

Zunächst wurde bereits erwähnt, dass ein Werkzeug Verfahren verwendet, um ein höheres Ziel zu erreichen. Das heißt, mit einem Werkzeug geben wir dem Benutzer eine Black-Box, der eine Anzahl von Einstellungen übergeben wird und anschließend gemäß seiner Bestimmung auf dem Datenbestand arbeitet. Ein Werkzeug beschäftigt sich dabei immer mit genau einer Aufgabe. Diese ist, eine klar definierte Bereinigung durchzuführen. Das bedeutet, dass für jede Art einer Bereinigung ein Werkzeug erforderlich ist. Die Umsetzung des Verfahrens innerhalb der Werkzeuge ist von der Aufgabe des Werkzeuges abhängig. Als Beispiel sei die Fallunterscheidung zwischen Tupel- und Attribut-Ebene im Abschnitt 4.5 erwähnt.

Aus Benutzersicht ist eine gewünschte Datenbereinigung eine Black-Box. In dieser Werkzeugsuite setzt sich eine solche Datenbereinigung aus einzelnen Werkzeugen zusammen, die jedes für sich eine Bereinigung ausführt. Das Ziel ist es, dem Benutzer zu ermöglichen, seine eigene Vorstellung einer Datenbereinigung zu konfigurieren, indem er die gewünschten Werkzeuge der Prozessliste hinzufügt. Wir geben dem Benutzer folglich eine White-Box zur Datenbereinigung an die Hand, die sich flexibel an seine Wünsche anpassen lässt. Klassische Zusammenstellungen von Werkzeugen können dem Benutzer in Form von wählbaren Voreinstellungen gegeben werden. Eine solche Zusammenstellung wäre zum Beispiel das Verwenden des Werkzeuges zum Korrigieren von Attributwert-Duplikaten vor dem Tupel-Duplikatbereiniger.

Die Werkzeuge, die im Framework verfügbar sein werden, sind zu unterteilen in

- Attributwerkzeuge
- Tupelwerkzeuge

Dabei befassen sich die Attributwerkzeuge mit der Korrektur von Datenwerten, also Attributwerten, während die Tupelwerkzeuge sich mit dem Korrigieren von Datenzeilen befassen. Dabei sei als Tupelwerkzeug hauptsächlich die Duplikaterkennung genannt. Dies geht darauf zurück, dass für die nach Abschnitt 1.2 genannte Aufgabe keine gesamten Tupel ausgetauscht werden brauchen. Zur Vollständigkeit sei aber eine Tupellöschoperation vorgestellt. Dahingegen sind eine Reihe spezieller Attributwerkzeuge zu entwickeln, wie sie in 2.5 bereits angesprochen wurden.

Für alle Werkzeuge gelten die Frameworkeinstellungen, welche die Tabellen und Attribute festlegen, auf denen die Werkzeuge agieren sollen. Die Werkzeugparameter sind Teil der Werkzeugeinstellungen und abhängig vom Werkzeug.

Zur Festlegung der Attribute zählt ebenfalls das Zuweisen von Attributeigenschaften. Dies ist, in Analogie zu der Definition des Begriffes im Abschnitt 2.1.1, der Attributname, Datentyp bzw. Dointyp und eventuell das Attributgewicht für Werkzeuge, die das PC-Filter Verfahren nutzen. Die Menge dieser Zuweisungen erzeugt die Liste der Attributeigenschaften. Der ursprüngliche Begriff wurde dadurch erweitert. Das Eingabeformat sei für die hier vorgestellten Werkzeuge nicht zur Wahl eines Ähnlichkeitsmaßes notwendig. Vielmehr stellt es eine Werkzeugeinstellung dar.

Die im Konfigurationsblock festgelegten Werkzeugeinstellungen werden hier zu den Parametern der Werkzeuge. Welche der Parameter auch Teil der Konfiguration sind, hängt von der Implementierung der Werkzeuge ab, da es möglich ist, manche Parameter vor-eingestellt zu lassen.

Die nachfolgenden Konzepte für Werkzeuge beschreiben die Werkzeugblöcke „Ausführung“, „Fehlererkennung“ und „Fehlerkorrektur“, die durch den Block „Prozess“ des Frameworks gestartet werden. Die Blöcke wurden beschrieben in den Abschnitten 4.2.2 und 4.3.

Ein Werkzeug hat dabei fünf beschreibende Eigenschaften:

1. Die Aufgabe des Werkzeuges ist dem Namen zu entnehmen und beschreibt die Bereinigung, die es ausführt.
2. Ein Werkzeug erhält vom Framework Elemente der Konfiguration als **Parameter** übergeben. Darunter auch dessen Werkzeugeinstellungen.
3. Erstellt ein Werkzeug ein neues, persistentes Objekt so erzeugt es eine **Ausgabe**. Dies kann zum Beispiel das Schaffen einer *View* auf der Datenbank sein. Auch der Rückgabewert einer Funktion kann eine Ausgabe sein.
4. Während der Abarbeitung eines Werkzeuges können temporäre Daten anfallen. Ist es dazu notwendig, aufgrund der Datenmenge kurzzeitig eine neue Tabelle anzulegen, so ist dies eine **temporäre Tabelle**.
5. Ein Werkzeug kann zu einer **Interaktion** auffordern, wenn es auf die Hilfe vom Benutzer angewiesen ist.

Parameter von Werkzeugen setzen sich zusammen aus den Parametern, die für die verwendeten Verfahren notwendig sind, sowie durch Werkzeugeinstellungen, die für die spezifische Ausführung des Werkzeuges notwendig sind. Zum Beispiel, welches Verfahren verwendet werden soll.

4.4.1 Tupelwerkzeuge

Werkzeuge dieser Kategorie arbeiten auf einer Menge von Tupeln. Die Attribute der Tupel spielen lediglich für die untergeordneten Verfahren eine Rolle.

4.4.1.1 Tupel-Duplikatbereiniger

Parameter: Menge an Tupeln, Liste der Attributeigenschaften, Anzahl Partitionen, Größe des Randbereichs, Genauigkeit zwischen $[0,1]$, zusätzliche Optionen für Merging (siehe 4.5)

Ausgabe: keine

Temporäre Tabellen: nach jedem Funktionsblock

Interaktion: Clusterkorrektur, Referenzwert

Wir wenden PC-Filter als Duplikaterkenner an. PC-Filter hat dabei fünf Vorteile:

- Es ist in einzelne Funktionsblöcke aufteilbar
- Eine Implementierung nur über Zuordnungsnummern ist möglich
- Verfahren zur Ähnlichkeitsfaktorbildung kann leicht getauscht werden
- Es lässt sich vollständig in PL/SQL implementieren
- Aus der Duplikattabelle (s.u.) lassen sich Cluster von Zuordnungsnummern ähnlicher Tupel bilden

Zunächst erlaubt das leichte Aufteilen in einzelne Funktionsblöcke zum Einen den Austausch eben Solcher, und zum Anderen bringt das Verfahren von Haus aus Unterbrechungspunkte mit, an denen temporäre Daten bereits physisch abgelegt wurden. Siehe dazu auch Abschnitt 4.8. Um von dieser Eigenschaft Gebrauch zu machen, unterteilen wir die drei Schritte des Verfahrens in kleinere Funktionsblöcke, nach denen jeweils ein fester Zustand erreicht wurde. Das Ergebnis der Erkennung wird in einer temporären Duplikattabelle gesichert.

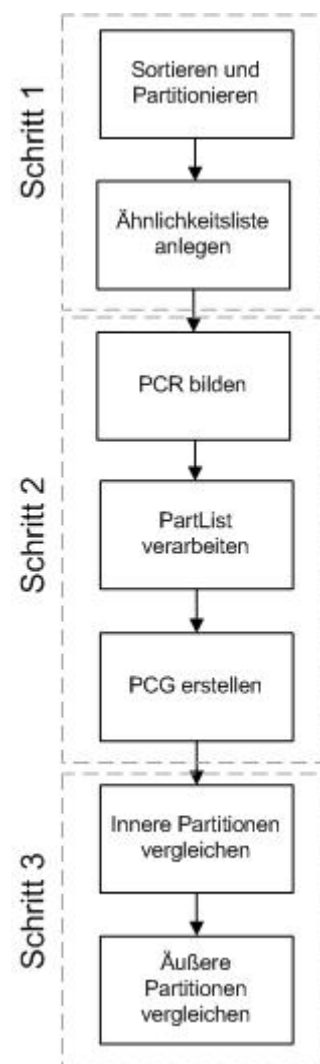


Abbildung 4.5: PC-Filter Funktionsblöcke

PC-Filter selbst bedarf zudem keinen Zugriff auf die Attributwerte. Alle PC-Filter Operationen sortieren lediglich Zuordnungsnummern. Der eigentliche Zugriff erfolgt über die Verfahren zur Erzeugung von Ähnlichkeitsfaktoren. Damit ist das Verfahren sehr variabel was unterschiedliche Datentypen oder Domänen betrifft. Es muss lediglich das Ähnlichkeitsmaß angepasst werden. Die Attribute, die PC-Filter zum Vergleich heranziehen soll, sind in der Liste der Attributeigenschaften abgelegt. Anhand dieser wird auch das Ähnlichkeitsmaß bestimmt.

Der Ablauf von PC-Filter erfolgt wie in 3.5 bereits vorgestellt. Das Verfahren ist vollständig in PL/SQL programmierbar. Damit ist es möglich, die Abfragen an den Datenbestand direkt auf der Datenbanke zu starten und die Schnittstellenzeit zu sparen. Zudem lassen sich interne Tabellenstrukturen für die Implementierung nutzen.

Betrachtet man ein Attribut als eine einspaltige Tabelle, so kann man diese ebenfalls mit PC-Filter behandeln. Damit ist es möglich, ähnliche Attributwerte zu finden und zu gruppieren (Siehe hierzu auch die Ähnlichkeit der Ablaufpläne im Abschnitt 2.2).

Wird PC-Filter auf Tabellen angewendet, muss davon ausgegangen werden, dass es Verweise in Form von Foreign-Keys geben kann. Dazu müssen alle Tabellen gefunden werden, die einen Foreign-Key auf die gewählte Tabelle haben und diese iterativ abgearbeitet werden. Das Herausfinden der Tabellen und das Vorbereiten des Werkzeuges ist Teil der Werkzeugeinstellung und beeinflusst den Ausführungsblock des Werkzeuges. Dieser veranlasst das wiederholte Ausführen der Blöcke der Verarbeitungsebene.

Nach Durchlaufen der Duplikaterkennung wird Merging (vgl. 4.5 auf die erzeugte temporäre Duplikattabelle angewendet.

Zuletzt werden die temporären Tabellen entfernt.

Wird die aktuelle Tabelle referenziert, so muss bei der Werkzeugeinstellung dieses Berücksichtigt werden. Die Schlüssel der verweisenden Tabellen müssen angepasst werden. Verweist die aktuelle Tabelle auf Tupel einer anderen Tabelle, so muss zunächst diese andere Tabelle bereinigt werden. Es gelten die Zusätze zum Ablaufplan aus 2.2.

4.4.1.2 Tupel-Entferner

Parameter: Tabelle, Suchbedingung

Ausgabe: keine

Temporäre Tabellen: keine

Interaktion: Rückfrage mit Liste zu löschender Tupel

Dieses Werkzeug ist ein wertunabhängiges Werkzeug, dass es ermöglicht, Tupel anhand einer Suchbedingung zu löschen. Aus Sicherheitsgründen werden die gefundenen Tupel dem Benutzer zur Interaktion vorgestellt, damit dieser notfalls abbrechen kann. Fehlererkennung ist hierbei die Filterung nach der Suchbedingung und der Rückfrage vor dem Löschen. Die Fehlerkorrektur ist die anschließende Entfernung der Tupel. Wird die aktuelle Tabelle referenziert, so muss bei der Werkzeugeinstellung dieses Berücksichtigt werden.

4.4.2 Attributwerkzeuge

Lediglich auf der Attribut-Ebene erfolgt Datenmanipulation (mit Ausnahme des Löschens von Tupeln). Daraus folgt bereits, dass es für jeden denkbaren Domaintyp von Attributwert ein Attributwerkzeug zur Bereinigung geben müsste. In den folgenden Abschnitten werden die für das Anwendungsgebiet nach Abschnitt 1.2 und 2.5 benötigten Attributwerkzeuge beschrieben.

4.4.2.1 Formatierungsprüfer

Parameter: Tabelle, Eingabeformat, Attribut

Ausgabe: keine

Temporäre Tabellen: keine

Interaktion: abschaltbare Rückfrage

Dieses Werkzeug verwendet das Eingabeformat oder Domänenregeln, um gemäß seiner Konfiguration abweichende Attributwerte anzupassen. Zugrunde liegen Verfahren, die einen Attributwert auf das Eingabeformat anpassen. So kann zum Beispiel ein Verfahren Eingabeformate behandeln, die sich auf die Textformatierung beziehen. Also zum Beispiel das Format „erster Buchstabe immer großschreiben“ oder „nur 5 Zeichen erlaubt“. Dies ist offenbar eine Umsetzung einer wertunabhängiger Fehlerkorrektur. Ein Beispiel für die Anwendbarkeit dieses Werkzeuges wurde bereits im Abschnitt 2.1.1 bei der Definition von wertunabhängigen Fehlern gegeben.

4.4.2.2 Ersetzer

Parameter: Tabelle, Attribut, Suchmuster, Ersetzung

Ausgabe: keine

Temporäre Tabellen: keine

Interaktion: abschaltbare Rückfrage

Hier werden in Attributwerten eines konfigurierten Attributs vorkommende Suchmuster gegen die konfigurierte Ersetzung ersetzt. Das Suchmuster kann dabei ein regulärer Ausdruck oder eine Teilzeichenkette sein. Das jeweils passende Verfahren wird durch die Konfiguration bestimmt. Damit ist es zum Beispiel möglich, alle 'Mustermann' in „Musterfrau“ zu ändern, aber auch komplexere Ersetzungen vorzunehmen. Auch hier handelt es sich um eine wertunabhängige Fehlerkorrektur.

4.4.2.3 Attributwert-Duplikatbereiniger

Parameter: Tabelle, Attribut, Attributeigenschaften, zusätzliche Optionen für Tupel-Duplikatbereiniger (siehe 4.4.1.1)

Ausgabe: keine

Temporäre Tabellen: gemäß Tupel-Duplikatbereiniger

Interaktion: gemäß Tupel-Duplikatbereiniger

Bei diesem Werkzeug handelt es sich um einen Attributwertvergleich. Es sollen dabei alle Attributwerte ähnlicher Schreibweise gruppiert werden und auf einen Referenzwert angepasst werden. Dies hat das Ziel, Schreibfehler zu beheben und ähnliche Schreibweisen zu normalisieren.

Der Aufbau der Fehlererkennung entspricht einer Vereinfachung des Tupel-Duplikatbereinigers.

Dazu wird zunächst das aus den Attributeigenschaften abgeleitete Ähnlichkeitsmaß gewählt und ein entsprechendes Vergleichsverfahren für PC-Filter vorbereitet. Anschließend wird das zu behandelnde Feld als eine einspaltige Tabelle durch PC-Filter analysiert.

Zur Fehlerkorrektur verwenden wir wieder das Merging nach 4.5. Durch das Übergeben von nur einem Attribut reduziert sich das Merging auf ein Referenzwertverfahren nach 4.6.

Der Ablauf dieses Attributwerkzeuges wurde auch im Abschnitt 2.2 grob beschrieben. Wir machen uns hier zunutze, dass Merging, beschrieben nach 4.5, unter Ausnutzung der Bedingungen aus 2.4.2, immer auch ein Referenzwertverfahren einleitet.

4.4.2.4 ISBN-Prüfer

Dieses Werkzeug ist Teil der aus Abschnitt 2.5 geforderten Attributwerkzeugen.

Parameter: Tabelle, Attribut, ISBN-Format

Ausgabe: keine

Temporäre Tabellen: ungültige Nummern

Interaktion: Benutzerkorrektur für unklare Fälle

Ziel des Werkzeuges ist es, falsch formatierte ISBNs zu korrigieren. Dazu behandelt es die Attributwerte eines Attributes als ISBN. Das Werkzeug wird in zwei Schritte durchgeführt:

Als Erstes wird auf Gültigkeit der ISBN hinsichtlich Stellenzahl und Zeichenfolge ohne Beachtung von Trennzeichen geprüft. Nicht gültige Nummern werden gesammelt und später zur Benutzerinteraktion angezeigt. Vorzugsweise mit weiteren Informationen für den Benutzer. Eine entsprechende Einstellung zur Darstellung kann in den Werkzeugeinstellungen konfiguriert werden.

Als Zweites wird versucht, das vorgegebene ISBN-Format anzuwenden. Dies kann zum Beispiel das Einsetzen von Trennzeichen sein oder die Umrechnung in 10 bzw. 13 stellige Nummern. Nicht berechenbare Nummern werden ebenso als ungültige Nummer abgelegt und später zur Benutzerinteraktion angezeigt.

4.4.2.5 Autoren-Formatbereiniger

Dieses Werkzeug ist Teil der aus Abschnitt 2.5 geforderten Attributwerkzeugen.

Parameter: Tabelle, Attribut, Format

Ausgabe: keine

Temporäre Tabellen: keine

Interaktion: Benutzerkorrektur für unklare Fälle

Ziel des Werkzeuges ist es, falsch formatierte Autoren-Einträge zu korrigieren. Dazu werden die Attributwerte mit dem Format verglichen. Das Format kann dabei zum Beispiel in Form eines regulären Ausdrucks vorliegen. Kann der Attributwert nicht mit dem Format vereinbart werden, so wird versucht, den Attributwert auf das Format abzuändern. Das kann erfolgen, indem im Attributwert nach dem im Format festgelegten Trennzeichen gesucht wird. Kann ein Trennzeichen zugeordnet werden, so kann davon die Position des Autornamens und von Leerzeichen bestimmt werden. Es wird also eine Art rückwärtiges Parsing verwendet, um in dem Attributwert Elemente des Formats zu bestimmen. Sobald diese Elemente bestimmt sind, können diese neu formatiert werden. So können zum Beispiel fehlende Leerzeichen vor Trennzeichen eingefügt werden oder ein Vertauschen von Vor- und Nachnamen aufgrund eines fehlenden Kommas erkannt werden.

4.4.2.6 Nullwert-Finder

Dieses Werkzeug ist Teil der aus Abschnitt 2.5 geforderten Attributwerkzeugen.

Parameter: Tabelle, Attribute

Ausgabe: keine

Temporäre Tabellen: keine

Interaktion: Liste mit Tupeln, dessen gewählte Attribute Nullwerte haben

Ziel des Werkzeuges ist es, dem Benutzer alle Tupel anzuzeigen, die in den gewählten Attributen Nullwerte haben.

Teil der Interaktion ist es, einem oder mehreren dieser Tupel für die Attribute Werte zu spezifizieren.

4.5 Mergingverfahren

Hier wird die Funktionsweise des Merging näher konzipiert. Das Merging ist ein Verfahren der Fehlerkorrektur und wird von Werkzeugen im Zusammenhang mit Duplikaterkennung auf Tupel-Ebene aufgerufen. Wie schon in 4.4 beschrieben, hat auch das Mergingverfahren beschreibende Eigenschaften:

Parameter: Duplikattabelle (siehe 4.4.1.1), Attributeigenschaften, Grenze für Clusterkorrektur, zusätzliche Optionen für Referenzwertverfahren (siehe 4.6)

Ausgabe: keine

Temporäre Tabellen: Ersetzungstabelle

Interaktion: Clusterkorrektur, Referenzwert

Beim Merging gehen wir von einem Cluster von mindestens zwei Tupeln aus. Dies ist bei Verbindung mit PC-Filter als Erkennungsverfahren bereits größtenteils geschehen. Es bedarf lediglich dem Filtern der jeweiligen Cluster aus der Duplikattabelle durch geschickte Selektierung. Der erste Schritt ist daher das Selektieren von Clustern. Wir gehen nun die Cluster durch und führen die folgenden Schritte aus:

Die Tupel des Clusters werden Attributweise durchlaufen. Für jedes Attribut wird das Referenzwertverfahren angewendet. Der Cluster ähnlicher Attributwerte für das Referenzwertverfahren ist dann die Attributwertmenge des Attributes. Die Umsetzung folgt der Darstellung in Abschnitt 2.4.2.

Ersetzungen werden in einer Ersetzungstabelle zwischengespeichert, um beim Feststellen eines Clusterfehlers reagieren zu können. Dies wird umgesetzt, indem die der Tupelcluster nochmals mit der Ersetzungstabelle verglichen wird. Zeigt ein Tupel zu große Unterschiede (übersteigt die gesetzte Grenze), so wird eine Benutzerinteraktion verlangt. Bei der Interaktion kann der Benutzer auch nachträglich noch Werte in der Ersetzungstabelle ändern. Die Eintragungen der Ersetzungstabelle ersetzen nach Durchlaufen aller Attribute die Werte der Originaltupel.

Wird das Merging als Teil des Tupel-Duplikatbereinigers gestartet, wird nach dem Übernehmen der Ersetzungstabelle ein Tupel bestimmt, das als Originaltupel fungieren soll. Dies kann beispielsweise Konfigurationsabhängig geschehen, nach Primärschlüssel oder sogar nach Füllstand der Attribute. Nach dem Durchlaufen der Attribute werden alle als nicht-original markierten Tupel gelöscht. Dieser Fall entfällt, wenn ein Werkzeug der Attribut-Ebene aktiv ist.

Wird eine Clusterkorrektur ausgelöst, so wird dem Benutzer der derzeitige Tupelcluster angezeigt. Der Benutzer kann dann Tupel aus dem Cluster entfernen. Diese Tupel werden dann nicht weiter behandelt, da der Fehlererkennungsblock bereits abgelaufen ist. Um diese ausgelassenen Tupel erneut zu gruppieren, muss ein neuer Erkennungsvorgang aus der Prozessliste gestartet werden. Nachdem der Cluster korrigiert wurde, wird obiger Vorgang mit dem neuen Cluster neugestartet. Es wurde also ein Prozessschritt verdeckt zurückgegangen.

4.6 Referenzwertverfahren

Hier wird die Funktionsweise des Referenzwertverfahrens näher konzipiert. In den hier vorgestellten Werkzeugen wird das Referenzwertverfahren als Teil des Merging (siehe 4.5) realisiert, indem die in 2.4.2 dargestellten Regeln umgesetzt werden. Wie schon in 4.4 beschrieben, hat auch das Referenzwertverfahren beschreibende Eigenschaften:

Parameter: Cluster von Attributwerten, Attributeigenschaften, Grenze für Automatik, optionale Referenzwertvoreinstellung

Ausgabe: Referenzwert

Temporäre Tabellen: keine

Interaktion: Referenzwert, abschaltbar Kontrolle vom Referenzwert für Voreinstellungsautomatik

Das Verfahren ist als Funktion umzusetzen, da dies mehr Flexibilität mit Rollback-Operationen erlaubt. So führt das Merging beispielsweise eine Ersetzungstabelle. Diese Eigenschaft bietet zudem wieder einen Unterbrechungspunkt.

Beim Referenzwertverfahren gehen wir von der Übergabe einer Menge von Attributwerten mit hoher Ähnlichkeit aus. Dies ist bereits der Fall, wenn das Verfahren durch Merging aufgerufen wurde. Im Weiteren folgen wir dem Vorgehen wie in Abschnitt 2.4.1 beschrieben.

Zunächst wird versucht, den Referenzwert anhand der Referenzwertvoreinstellung zu ermitteln. Dazu werden die voreingestellten Referenzwerte mit allen Attributwerten verglichen. Wird eine deutliche Ähnlichkeit zu den meisten Werten festgestellt, so wird dieser Referenzwert übernommen.

Anschließend wird versucht, durch das mehrheitliche Vorkommen eines Wertes überhalb der gesetzten Grenze den Referenzwert zu bestimmen.

Schlägt auch dies fehl wird eine Benutzerinteraktion verlangt, bei dem der Benutzer den Referenzwert zu den Attributwerten wählen kann. Befindet sich der Nutzer in Duplikatbereinigerwerkzeugen, sollte er ebenfalls eine Clusterkorrektur fordern können, da er eventuell frühzeitig in der Lage ist, einen solchen Erkennungsfehler festzustellen.

4.7 Ähnlichkeitsverfahren

In diesem Abschnitt behandeln wir die zur Ähnlichkeitsfaktorherzeugung notwendigen Ähnlichkeitsmaße bzw. direkt die erzeugenden Funktionen. Die Ähnlichkeitsmaße sind ein Hauptbestandteil der Duplikaterkennung mit PC-Filter. Daher erfüllen die vorgestellten Maße die in Abschnitt 3.5 beschriebene Dreiecksungleichung. Zur Erstellung eines Ähnlichkeitsfaktors werden wir zunächst aus, auf welcher Ebene das Vergleichsverfahren gestartet wurde. Anschließend bestimmen wir aus den Attributeigenschaften das entsprechende Ähnlichkeitsmaß. Der Faktor wird dann aus einer vorgeschlagenen Funktion gebildet.

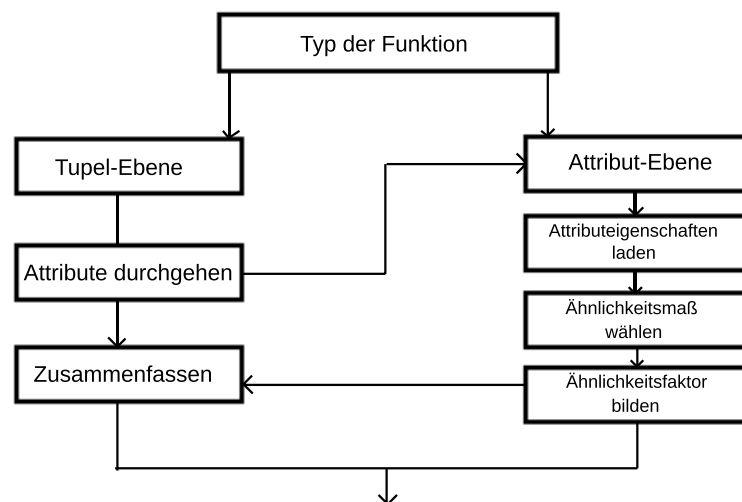


Abbildung 4.6: Vergleichsverfahren Ablaufdiagramm

Alle hier vorgestellten Ähnlichkeitsmaße und Faktorherzeuger sind Verfahren, die von Werkzeugen aufgerufen werden können. Demnach haben diese Verfahren auch beschreibende Eigenschaften, die denen aus 4.4 ähneln. Alle hier vorgestellten Verfahren sind als Funktionen umzusetzen. Deren **Ausgabe** ist gemäß Definition ein Wert zwischen 0 und 1. Eine **Benutzerinteraktion** ist nicht notwendig. **Temporäre Tabellen** können durch interne Datenstrukturen ersetzt werden, da die Verfahren keine Unterbrechungspunkte bieten und der Datenaufwand gering ist.

4.7.1 Boolescher Vergleich

Parameter: zwei beliebige Attributwerte

Die einfachste Form eines Ähnlichkeitsmaßes ist der direkte Vergleich zweier Werte auf Gleichheit. Das Ergebnis ergibt einen Ähnlichkeitsfaktor von 0 oder 1.

4.7.2 Numerischer Vergleich

Parameter: zwei Attributwerte eines numerischen Datentyps

Bei dem Vergleich von numerischen Werten können drei Ansätze gewählt werden:

1. Der Wert ist eindeutig.
2. Verhalten wie Zeichenfolge (siehe auch Hinweis in 2.3.1)
3. Dynamisches Vorgehen

Fall 1 impliziert den booleschen Vergleich. Im Fall 2 kann Q-Grams mit $q=2$ angewendet werden. Field-Similarity ist unpraktisch, da Zahlen mit den gleichen Ziffern unter Umständen trotz starker Unähnlichkeit mit einem Faktor 1 erkannt werden würden.

Für diese Konzeption schlagen wir daher das Entwickeln eines dynamischen Vorgehens vor. Wir gehen von zwei möglichen Fehlerfällen aus: Zahlendreher und vergessene Ziffer. Die Rückgabe ist aus der Menge $\{0 ; 0,7 ; 1\}$. Die Wahl des mittleren Elements ist dabei willkürlich und sollte knapp oberhalb der Schranke für hohe Ähnlichkeit liegen. Ziel ist es, eine Ziffernfolge als ungleich, identisch oder eventuell ähnlich zu bewerten.

Zunächst vergleichen wir die Zahlen mit dem booleschen Vergleich. Ist der ermittelte Faktor 0 können wir den Betrag der Differenz bilden. Ist dieser unter 9 mit Ausnahme der 0 kann kein Zahlendreher vorliegen, zudem kann Zahl A nicht Teilstück von Zahl B sein.

$$|01 - 10| = 9, |54 - 45| = 9, |1 - 10| = 9, |5 - 55| = 50$$

Beispiel 4.2: Beispiel für Differenzbeträge

Trifft dies nicht zu, vergleichen wir die Anzahl der Ziffern. Ist diese ungleich, so ist es recht unwahrscheinlich, dass ein Zahlendreher vorliegt. Andernfalls können wir mit einem Sliding-Window der Größe 2 beide Zahlen parallel durchlaufen und auf Zahlendreher prüfen. Ist der Unterschied zwischen der Anzahl der Stellen der Größe 1 können wir prüfen ob Zahl A Teil von Zahl B ist. In diesem Fall ist es wahrscheinlich, dass eine Ziffer bei der Eingabe übersehen wurde.

Erkennt der boolesche Vergleich die Identität, geben wir 1 zurück. Bei der Erkennung von Zahlendreher oder vergessener Ziffer geben wir 0,7 als Wert für eventuelle Ähnlichkeit zurück. Ansonsten geben wir 0 zurück.

4.7.3 Datums-Vergleich

Parameter: zwei Attributwerte vom Datentyp Datum, Vergleichseinheit, Vergleichsart

Dieses Maß bestimmt die Ähnlichkeit zwischen zwei Oracle Datumswerten. Dabei ist vorzugeben, auf welche kleinste Einheit (Jahr, Monat, Tag, ...) geprüft werden soll. Je nach Konfiguration bzw. Einsatzes kann dieses Maß entweder die Gleichheit von Daten feststellen oder auf Zahlendreher in den Zeitbereichen prüfen. Dazu können ähnliche Verfahren wie beim numerischen Vergleich verwendet werden.

4.7.4 Modifizierte Field-Similarity

Parameter: zwei Attributwerte eines als Zeichenfolge darstellbaren Datentyps, Groß-/Kleinschreibung beachten

Ziel dieser Modifikation ist es, die Verfahren Field-Similarity, Q-Grams und das Prinzip des Atomic-Strings zu kombinieren. Dies soll es ermöglichen, sowohl Zeichendreher als auch unähnliche Zeichenfolgen mit gleichem Zeichenvorrat unterscheiden zu können. Ausserdem soll es uns ermöglichen, kurze Sätze oder Namen-Vornamen-Kombinationen vergleichen zu können.

Atomic-Strings wurde bereits in Abschnitt 2.3.1 erwähnt. Hier wird dessen Grundlage verwendet, um ein eigenes Verfahren zu schaffen. Der Ablauf von Q-Grams wurde in Abschnitt 3.2 erläutert. Field-Similarity ist zu definieren wie in Abschnitt 3.3.

Der Grundschrift sieht vor, die Zeichenketten in deren Mengen von Atomic-Strings, also einzelne Wörter, zu zerlegen und die Gesamtzahl der Wörter beider Mengen zu bestimmen. Diese einzelnen Wörter werden untereinander mit einem boolschen Vergleich geprüft. Identische Wörter werden aus beiden Mengen entfernt und gezählt.

Wir erhalten die Anzahl identischer Wörter, die Gesamtzahl aller Wörter und zwei Mengen an Wörtern, die nun auf Ähnlichkeit geprüft werden müssen.

Die übrigen Wörter werden mit Field-Similarity geprüft und für jedes Wort dessen höchster Ähnlichkeitsfaktor gemerkt.

Alle Ähnlichkeitsfaktoren, die nicht auf eine Identität hinweisen, also unterhalb der höchstmöglichen Ähnlichkeit sind, werden zu der Anzahl identischer Wörter addiert und die zugehörigen Wörter aus ihren Mengen entfernt.

Die übrigen Wörter wurden durch den boolschen Vergleich als nicht identisch gewertet, aber wurden durch Field-Similarity entsprechend bewertet. Dies lässt auf einen gleichen Zeichenvorrat schließen, bei dem Field-Similarity die Reihenfolge des Auftretens gemäß dessen Definition aus Abschnitt 3.3 ignoriert. Auf diese Wörter wird nun Q-Grams mit $q=2$ angewendet und erneut der höchste Ähnlichkeitsfaktor gemerkt.

Alle Ähnlichkeitsfaktoren werden nun mit dem höchstmöglichen Ähnlichkeitsfaktor addiert, durch 2 geteilt und anschließend zu der Anzahl identischer Wörter addiert. Dadurch wird ein Mittelweg gewählt zwischen den beiden Verfahren, indem der Mittelwert beider aus der Q-Grams-Bewertung mit der Field-Similarity-Bewertung gebildet wird. Letzterer Wert entspricht, durch die Bedingung aus dem vorangegangenen Schrittes, dem höchsten Ähnlichkeitsfaktor der gebildet werden kann. Verdeutlicht wird später dies in Beispiel 4.4.

Die Anzahl identischer Wörter, einschließlich der zuvor ermittelten Ähnlichkeitsfaktoren, wird nun durch die Gesamtzahl der Wörter geteilt. Wir bekommen die mittlere Ähnlichkeit aller Wörter der Zeichenfolgen. Die Zeichenfolgen lassen sich dann durch den Ähnlichkeitsfaktor bewerten, den die genannte Division liefert.

Beispiele:

- $A = \text{„Ich laufe gerne im nahen Wald“}$
 $B = \text{„Ich sehe gerne den Wald“}$
 $T_A = \{\text{Ich, laufe, gerne, im, nahen, Wald}\}$
 $T_B = \{\text{Ich, sehe, gerne, den, Wald}\}$
 $K = |T_A| + |T_B| = 11$
 Nach boolschen Vergleich:
 $T_A = \{\text{laufe, im, nahen}\}$
 $T_B = \{\text{sehe, den}\}$
 Wörter mit identischen Gegenstücken: $I = 6$
 Ähnlichkeitsfaktoren nach Field-Similarity:
 $(\text{„laufe“}, \frac{1}{5})$
 $(\text{„im“}, 0)$
 $(\text{„nahen“}, \frac{2}{5})$
 $(\text{„sehe“}, \frac{2}{5})$
 $(\text{„den“}, \frac{2}{5})$
 $I = 6 + \frac{1}{5} + 0 + 3 * \frac{2}{5} = 7,4$
 $\frac{I}{K} = \frac{7,4}{11} \doteq 0,67$

Beispiel 4.3: Modifizierte Field-Similarity für Sätze

- $A = \text{„Hanna“}$
 $B = \text{„Hanan“}$
 $K = 2$
 Nach boolschen Vergleich:
 $A = \text{„Hanna“}$
 $B = \text{„Hanan“}$
 Wörter mit identischen Gegenstücken: $I = 0$
 Ähnlichkeitsfaktoren nach Field-Similarity:
 $(\text{„Hanna“}, 1)$
 $(\text{„Hanan“}, 1)$
 $I = 0$
 nach Anwenden von $q=2$ auf alle Wörter mit Ähnlichkeitsfaktor 1:
 $(\text{„Hanna“}, \frac{1}{2})$
 $(\text{„Hanan“}, \frac{1}{2})$
 $I = 0 + \frac{\frac{1}{2}+1}{2} + \frac{\frac{1}{2}+1}{2} = 0 + 0,75 + 0,75 = 1,5$
 $\frac{I}{K} = \frac{1,5}{2} \doteq 0,75$
 $0,5 < 0,75 < 1$ (vgl. 3.4)

Beispiel 4.4: Modifizierte Field-Similarity für Zeichendreher

- $A = \text{“Hans Muster“}$
 $B = \text{“Muster, Hans“}$
 $K = 4$
 Nach booleschen Vergleich:
 $A = \text{“Muster“}$
 $B = \text{“Muster,“}$
 Wörter mit identischen Gegenstücken: $I = 2$
 Ähnlichkeitsfaktoren nach Field-Similarity:
 $(\text{„Muster“, } \frac{6}{7})$
 $(\text{„Muster,“, } \frac{6}{7})$
 $I = 2 + 2 * \frac{6}{7} = \frac{26}{7}$
 $\frac{I}{K} = \frac{\frac{26}{7}}{4} \doteq 0,93$

Beispiel 4.5: Modifizierte Field-Similarity für Name-Vorname-Kombinationen

- $A = \text{“Heinz“}$
 $B = \text{“Hinze“}$
 $K = 2$
 Nach booleschen Vergleich:
 $A = \text{“Heinz“}$
 $B = \text{“Hinze,“}$
 Wörter mit identischen Gegenstücken: $I = 0$
 Ähnlichkeitsfaktoren nach Field-Similarity:
 $(\text{„Heinz“, } 1)$
 $(\text{„Hinze,“, } 1)$
 nach Anwenden von $q=2$ auf alle Wörter mit Ähnlichkeitsfaktor 1:
 $(\text{„Heinz“, } \frac{1}{5})$
 $(\text{„Hinze,“, } \frac{1}{5})$
 $I = 0 + \frac{\frac{1}{5}+1}{2} + \frac{\frac{1}{5}+1}{2} = 0 + 0,6 + 0,6 = 1,2$
 $\frac{I}{K} = \frac{1,2}{2} \doteq 0,6$

Beispiel 4.6: Modifizierte Field-Similarity bei gleichem Zeichenvorrat

Field-Similarity auf die Eingabezeichenketten des Beispiels 4.5 angewendet führt zu einem Faktor von $\doteq 0,916$. Da $0,916 < 0,93$ liefert das modifizierte Verfahren eine bessere Erkennung der Identität der Aussage auf Ebene der Domäne beider Zeichenketten (vgl. 3.3).

Anhand des Beispiels 4.6 können wir sehen, dass mit dem modifizierten Verfahren unähnliche Zeichenketten mit gleichem Zeichenvorrat nicht mehr als identisch bewertet werden. Zudem zeichnet sich durch die Beispiele eine minimale Schranke ab, ab dem zwei Zeichenfolgen als ähnlich gelten. Diese Schranke ist ungefähr $\frac{2}{3}$.

```

Für jede Zeichenfolge
  Erstelle anhand eines Trennzeichens Menge von einzelnen Wörtern
K := Anzahl Wörter beider Mengen
I := 0
Für jedes Wort der Menge A
  Für jedes Wort der Menge B
    Wende boolschen Vergleich auf die Wörter an
    Wenn Vergleich hohe Ähnlichkeit liefert
      I := I + 2
      Entferne aktuelles Wort der Menge A
      Entferne aktuelles Wort der Menge B
Für jedes Wort der Menge A
  Für jedes Wort der Menge B
    Wende Field-Similarity auf die Wörter an
    Merke höchsten ermittelten Ähnlichkeitsfaktor für jedes Wort
Für alle Wörter
  Wenn Ähnlichkeitsfaktor < 1
    I := I + Ähnlichkeitsfaktor
    Entferne aktuelles Wort aus seiner Menge
Für jedes Wort der Menge A
  Für jedes Wort der Menge B
    Wende Q-Grams  $q=2$  auf die Wörter an
    Merke höchsten ermittelten Ähnlichkeitsfaktor für jedes Wort
Für alle Wörter
  I := I + Ähnlichkeitsfaktor
Ergebnis :=  $\frac{I}{K}$ 

```

Auflistung 4.1: Zusammenfassung der Modifizierte Field-Similarity

4.7.5 ISBN Vergleich

Parameter: zwei Attributwerte vom Domäntyp „ISBN“

Dieses Verfahren ist abgeleitet aus den, in 2.5 dargestellten, Werkzeugvorschlägen.

Dieses Verfahren verwendet einen boolschen Vergleich. Die Besonderheit besteht darin, dass es bei ISBN Werten verschiedene valide Eingabeformate gibt.

Vor dem boolschen Vergleich werden die Attributwerte auf ein Normformat gebracht. Am einfachsten ist hierbei das Weglassen von Trennzeichen, und die Umrechnung in das 13 stellige Format. Der boolsche Vergleich wird angewendet, da die Eintragungen der ISBN selten invalide sind. Die ISBN kann damit direkt als Kriterium zum Unterscheiden zweier Schriften genutzt werden. Dieses Maß ist nötig, da es häufig vorkommt, dass die ISBN in verschiedenen Eingabeformaten eingetragen wurde. Wurde zuvor nicht der ISBN-Prüfer angewendet kann daher nicht direkt mit dem boolschen Vergleich gearbeitet werden.

4.8 Prozessunterbrechung

Eine gewünschte Eigenschaft ist, den Bereinigungsprozess unterbrechen zu können, um ihn später erneut wieder aufzunehmen. Dazu müssen folgende Bedingungen erfüllt werden:

- Die aktuelle Position der Abarbeitung muss bekannt sein (Prozesszustand)
- Werkzeuge müssen unterbrechenbar sein
- Temporäre Daten, die ein Werkzeug benötigt, müssen persistent gespeichert sein

Die erste Bedingung wird dadurch erfüllt, dass die Abarbeitung der Werkzeuge der Prozessliste folgt und zudem der „*Controller*“ so konzeptioniert ist, dass jede „*Model*“-Operation einzeln angestoßen werden muss. Dadurch lässt sich der genau Zustand protokollieren. Dies wird mit dem Führen des Prozesszustandes wie in Abschnitt 4.3 erreicht.

Nach Erhalten des Pause-Signals muss der Prozessblock die Abarbeitung der Prozessliste bzw. des aktuellen Werkzeuges unterbrechen. Eine Unterbrechung kann nur stattfinden, wo kein Zugriff auf Daten erfolgt. Das heißt, es kann erst unterbrochen werden, wenn die letzte Anfrage an das „*Model*“ quittiert wurde. Um Abbrechen zu können, muss ein Werkzeug folglich in Detailblöcke (kleinste Funktionsblöcke) aufgeteilt werden, nach denen jeweils eine Unterbrechung durchgeführt werden kann. An diesen *Unterbrechungspunkten* müssen alle temporären Daten persistent Verfügbar oder Teil des Prozesszustandes sein. Dies erfüllt zugleich die letzte Bedingung, dass temporäre Daten gesichert werden. Dies ist notwendig, um in der Mitte unterbrochene Werkzeuge wieder fortsetzen zu können, auch dann, wenn lokale Prozedurvariablen bereits gelöscht wurden.

Das Ablegen des Prozesszustandes und der zugehörigen Konfiguration erfolgt wie in Abschnitt 4.2.3 beschrieben.

Durch die Verbindung des „*Model*“ mit dem DBS bietet es sich an temporäre Daten der Werkzeuge in Form von *temporären Tabellen* in der Datenbank abzulegen. Dabei ist darauf zu achten, einen Namen für die Tabelle zu wählen, der mit der Datenstruktur vereinbar ist. Das heißt, der Name ist nicht bereits Teil des Schemas.

Kapitel 5

Benutzerhandbuch

Dieses Kapitel soll den Umgang mit der Anwendung demonstrieren.

5.1 Installation und Ausführung

Die Anwendung besteht aus einem Packet mit drei Softwareteilen und einer Unterstützungsdatei. Das Hauptprogramm befindet sich in der *DBCcleaning.jar*. Als externe Bibliothek benötigt es den JDOM Parser aus der *jdom.jar* zum Verarbeiten von *XML* Dateien. Ergänzend zu dieser Datei gehört der JDBC Datenbanktreiber für Oracle 10 Datenbanken zum Packet. Dieser befindet sich in der Datei *ojdbc14_g.jar*. Schließlich gehört zu dem Packet noch eine *ZIP* Datei mit den Oracle OCI Instant Client Dateien für Windows und Linux Systeme in der passenden Version zum mitgelieferten JDBC Treiber.

5.1.1 Anforderungen

Für die Ausführung der Anwendung müssen folgende Anforderungen eingehalten werden:

- *ojdbc14_g.jar* muss im gleichen Verzeichnis mit dem Hauptprogramm liegen.
- *jdom.jar* muss im gleichen Verzeichnis mit dem Hauptprogramm liegen.
- Es müssen Oracle Client Dateien auf dem Client PC zur Verfügung stehen.
- Die installierten Oracle Client Dateien müssen mit dem JDBC Treiber kompatibel sein.
- Stehen keine Oracle Client Dateien zur Verfügung, kann der OCI Instant Client in Version 10 verwendet werden.
- Es wird eine Java Version ab 1.5.0_17 benötigt.

5.1.2 Vorbereitungen zur Installation

Kopieren Sie die drei *JAR* Dateien in ein gemeinsames Verzeichnis. Wenn Sie keinen installierten Oracle Client haben oder Sie keinen OCI Instant Client zur Hand haben, entpacken Sie die im Packet enthaltene *ZIP* Datei in dasselbe Verzeichnis, in das Sie die *JAR* Dateien kopiert haben. In ihr befindet sich der OCI Instant Client in der Version 10 für Windows 32Bit und Linux x86.

5.1.3 Ausführung

Um die Anwendung zu starten, führen Sie über eine GUI, zum Beispiel den Windows Explorer oder den Konqueror, die *DBCleaning.jar* aus. Auf der Kommandozeile können Sie das Programm starten mit:

```
java -jar DBCleaning.jar
```

Auflistung 5.1: Aufruf auf Kommandozeile

5.2 Hauptoberfläche: Konfiguration

Nach erfolgreichem Start sehen Sie die Hauptoberfläche auf der Registerkarte „Konfiguration“.

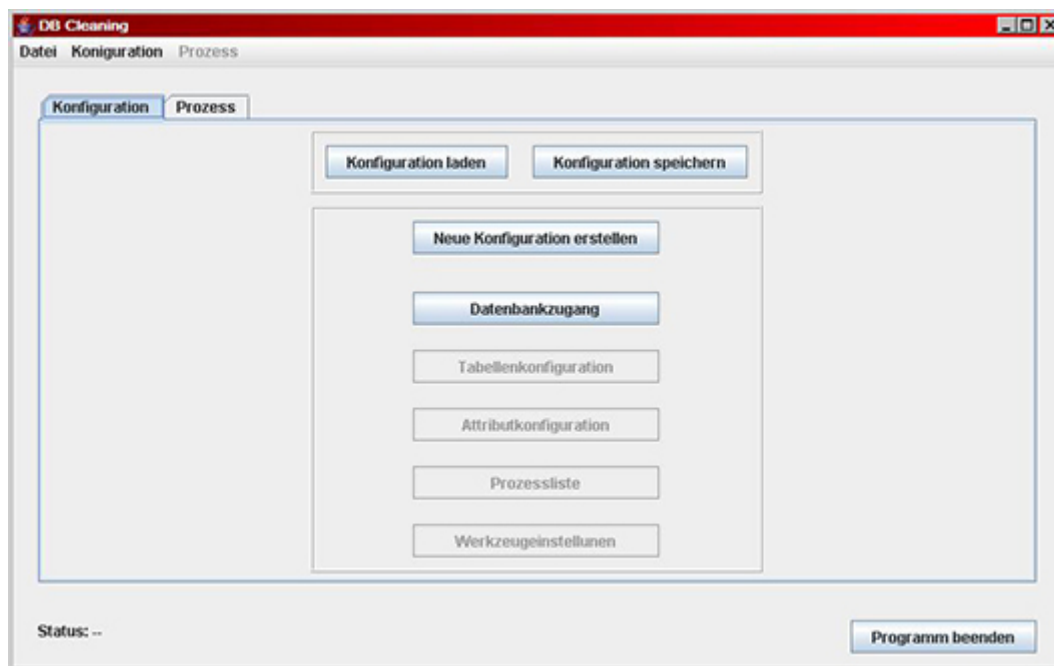


Abbildung 5.1: Hauptoberfläche: Konfiguration

Auf dieser Seite können Sie die Konfiguration verwalten.

Sie können zuvor erstellte Konfigurationen laden, indem Sie auf die Schaltfläche „Konfiguration laden“ klicken. Wollen Sie stattdessen Ihr Konfiguration für eine spätere Wartung sichern so betätigen Sie die Schaltfläche „Konfiguration speichern“. Beide Schaltflächen sind auch im Menü „Datei“ vorhanden.

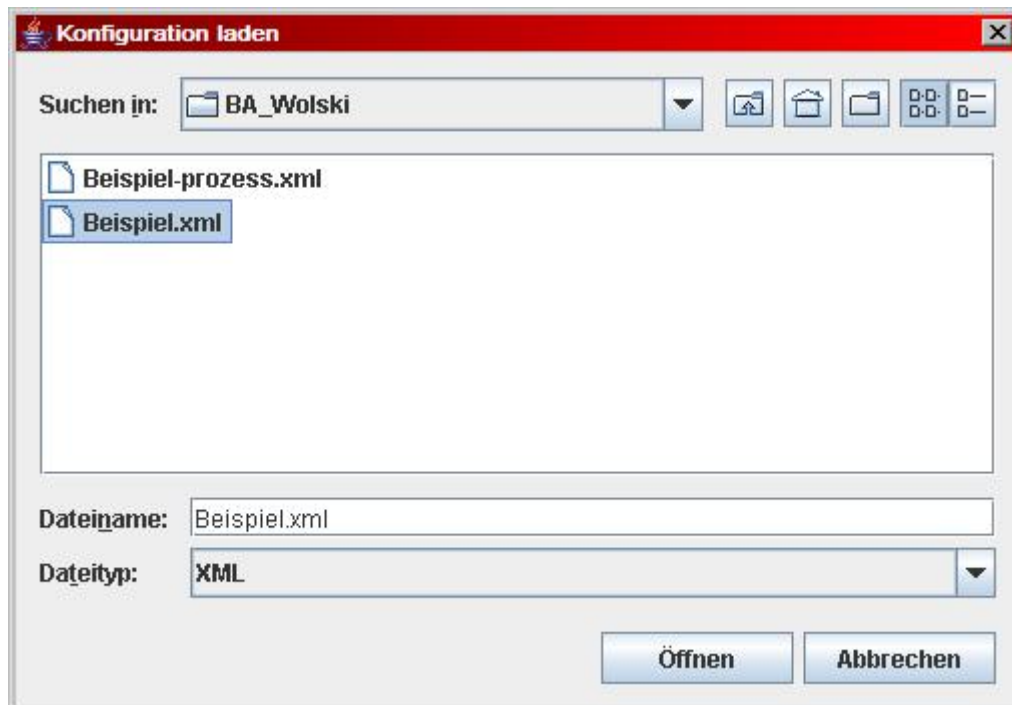


Abbildung 5.2: Konfiguration laden

Den Zustand der aktuellen Konfiguration können Sie im Hinweisfeld ablesen.

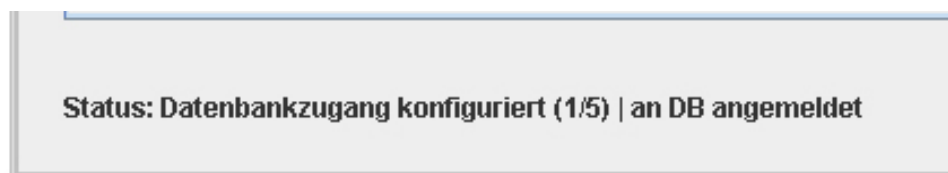


Abbildung 5.3: Hinweisfeld

Um die aktuelle Konfiguration zu entladen und eine neue Konfiguration zu erstellen klicken Sie auf „Neue Konfiguration erstellen“.

Sie können mit den Schaltflächen des Panels „Konfiguration“ zu den einzelnen Konfigurationsschritten springen. Achten Sie darauf, dass einige Konfigurationsabschnitte erst aktiv werden, wenn die vorhergehenden Konfigurationsschritte erfolgt sind. Ihnen stehen zur Auswahl:

- Datenbankzugang
- Tabellenkonfiguration
- Attributkonfiguration

- Prozessliste
- Werkzeugeinstellungen

Diese Schaltflächen sind auch im Menü „Konfiguration“ verfügbar.

Beachten Sie, dass eine Änderung der Konfiguration automatisch den Prozesszustand (siehe 5.3) zurücksetzt.

Sollten Sie eine Schaltfläche betätigen und folgenden Dialog sehen, so bedeutet dies, dass die Anwendung zurzeit nicht an der Datenbank angemeldet ist. Geben Sie Ihre Datenbanknutzerkennung ein, um die Anwendung mit der Datenbank zu verbinden.



Abbildung 5.4: Datenbanklogin

5.2.1 Datenbankzugang

Hier tragen Sie die Verbindungsdaten zu Ihrem Oracle Datenbank Server ein. In das Feld mit der Bezeichnung „Hostname“ tragen Sie die Adresse des Servers ein. Das Feld mit der Bezeichnung „Port“ ist für den Verbindungsport zuständig. Der Standardport für Oracle Datenbanken ist *1521*. In das Feld mit der Bezeichnung „SID“ tragen Sie das Schema ein, mit dem Sie verbinden wollen. Bitte wählen Sie ein Schema, in dem Ihr Datenbankbenutzer die Rechte „Select, Insert, Update, Create, Drop, Delete“ hat.



Abbildung 5.5: Datenbankzugang

Mit der Schaltfläche „Speichern“ werden Ihre Änderungen in die Konfiguration übernommen.

Mit der Schaltfläche „Speichern und weiter“ gelangen Sie zur Tabellenkonfiguration (5.2.2)

Mit der Schaltfläche „Abbrechen“ werden Ihre Änderungen ignoriert und das Fenster geschlossen.

5.2.2 Tabellenkonfiguration

In diesem Dialog wählen Sie die Tabellen, auf denen Sie Werkzeuge anwenden wollen.

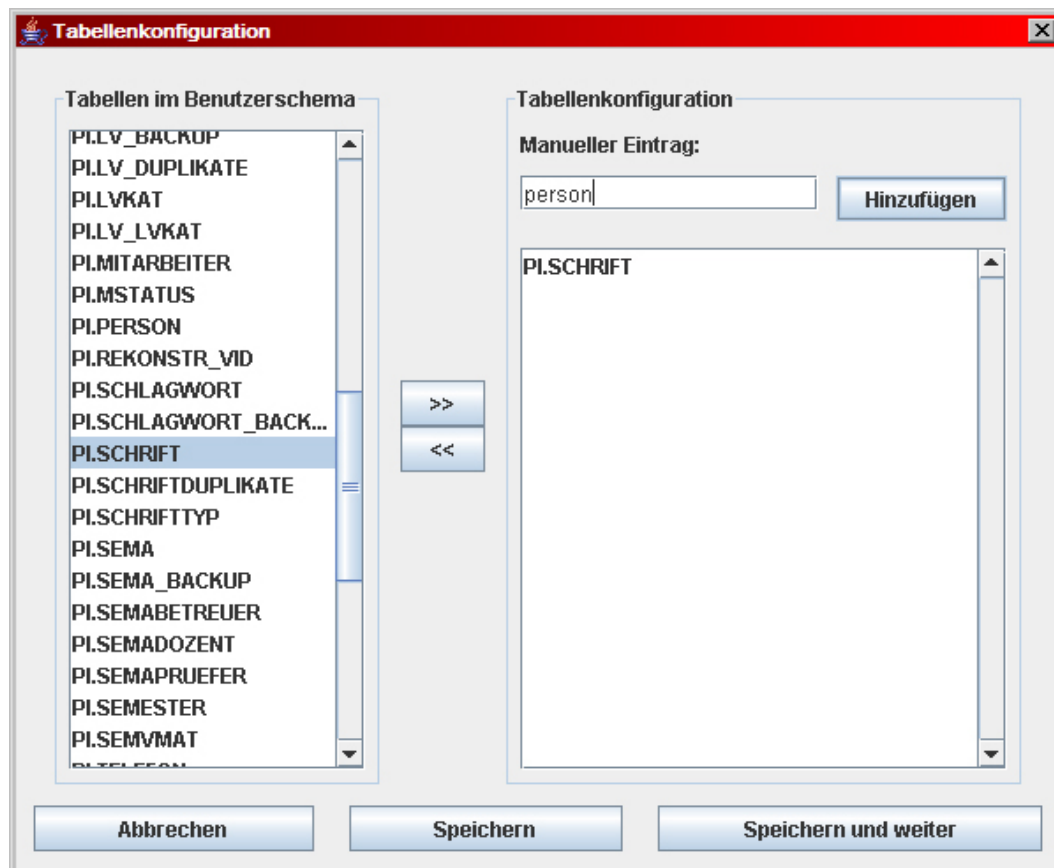


Abbildung 5.6: Tabellenkonfiguration

Sie können aus der automatisch ermittelten Liste von Tabellen mit der „>>“ Schaltfläche zuvor markierte Einträge in Ihre Konfiguration übertragen.

Wollen Sie eine Tabelle aus der Konfiguration entfernen, klicken Sie in der Liste im Panel „Tabellenkonfiguration“ den gewünschten Eintrag an und betätigen Sie die „<<“ Schaltfläche.

Um manuell eine Tabelle hinzuzufügen geben Sie den gewünschten Tabellennamen in das Feld mit der Beschriftung „Manueller Eintrag“ ein und betätigen Sie die „Hinzufügen“ Schaltfläche. Beachten Sie, dass vor dem Hinzufügen geprüft wird, ob die Tabelle für die Anwendung verfügbar ist.

Mit der Schaltfläche „Speichern“ werden Ihre Änderungen in die Konfiguration übernommen.

Mit der Schaltfläche „Speichern und weiter“ gelangen Sie zur Attributkonfiguration (5.2.3)

Mit der Schaltfläche „Abbrechen“ werden Ihre Änderungen ignoriert und das Fenster geschlossen.

5.2.3 Attributkonfiguration

In diesem Dialog legen Sie für die Attribute der gewählten Tabellen den Domaintyp fest. Standardmäßig wird der Datentyp des Attributes voreingetragen.

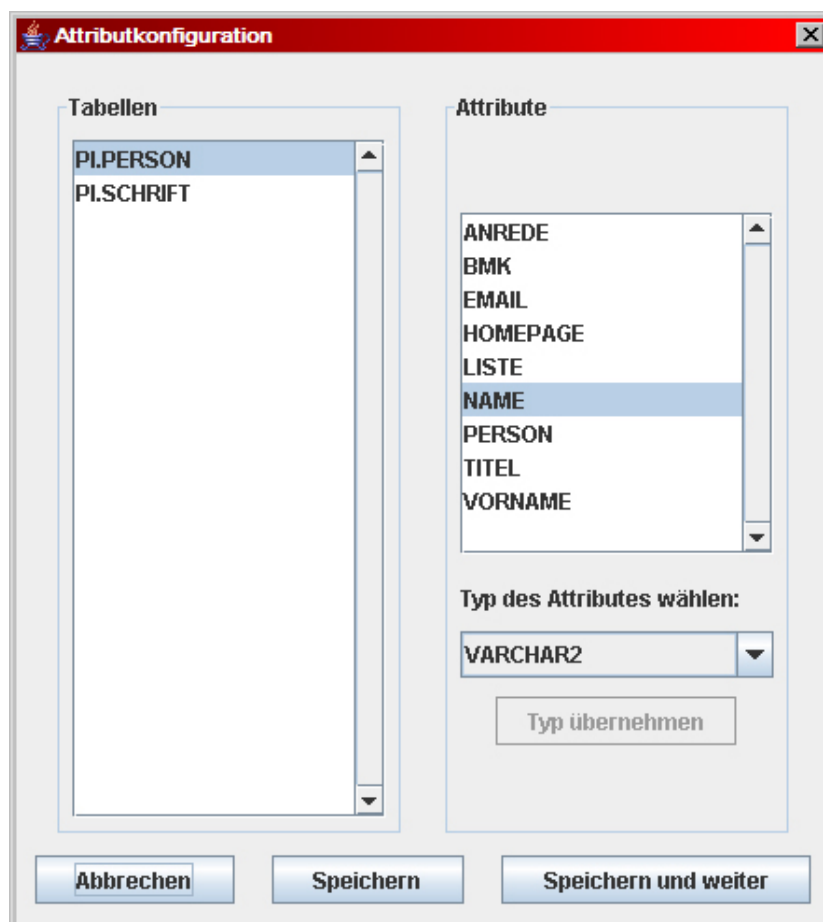


Abbildung 5.7: Attributkonfiguration

In der linken Liste können Sie eine Tabelle auswählen, im rechten Panel werden Ihnen anschließend die verfügbaren Attribute angezeigt. Klicken Sie auf ein Attribut um den zugewiesenen Daten- oder Domaintyp zu ändern. Die Typen können Sie aus der Combo-Box auswählen. Beachten Sie, dass Sie nach Ändern des Typs die Änderung mit der Schaltfläche „Typ Übernehmen“ bestätigen müssen.

Mit der Schaltfläche „Speichern“ werden Ihre Änderungen in die Konfiguration übernommen.

Mit der Schaltfläche „Speichern und weiter“ gelangen Sie zur Prozessliste (5.2.4)

Mit der Schaltfläche „Abbrechen“ werden Ihre Änderungen ignoriert und das Fenster geschlossen.

5.2.4 Prozessliste

Dieser Dialog dient der Verwaltung Ihrer Werkzeuge. Im linken Panel sehen Sie die verfügbaren Werkzeuge. Um ein Werkzeug der Prozessliste hinzuzufügen, markieren Sie das gewünschte Werkzeug im linken Panel und betätigen Sie die Schaltfläche „Hinzufügen“. Über die Schaltfläche „Voreinstellung“ können Sie typische Werkzeugkombinationen hinzufügen lassen.

Um ein Werkzeug wieder zu entfernen, markieren Sie im rechten Panel das Werkzeug und klicken Sie auf die Schaltfläche „Entfernen“.

Die Position des Werkzeuges in der Liste des rechten Panels ist ebenfalls die Startposition des Werkzeuges. Um ein Werkzeug früher zu starten, benutzen sie die Schaltflächen „Hoch“ und „Runter“ indem Sie rechts das Werkzeug markieren und dann auf die gewünschte Schaltfläche klicken.

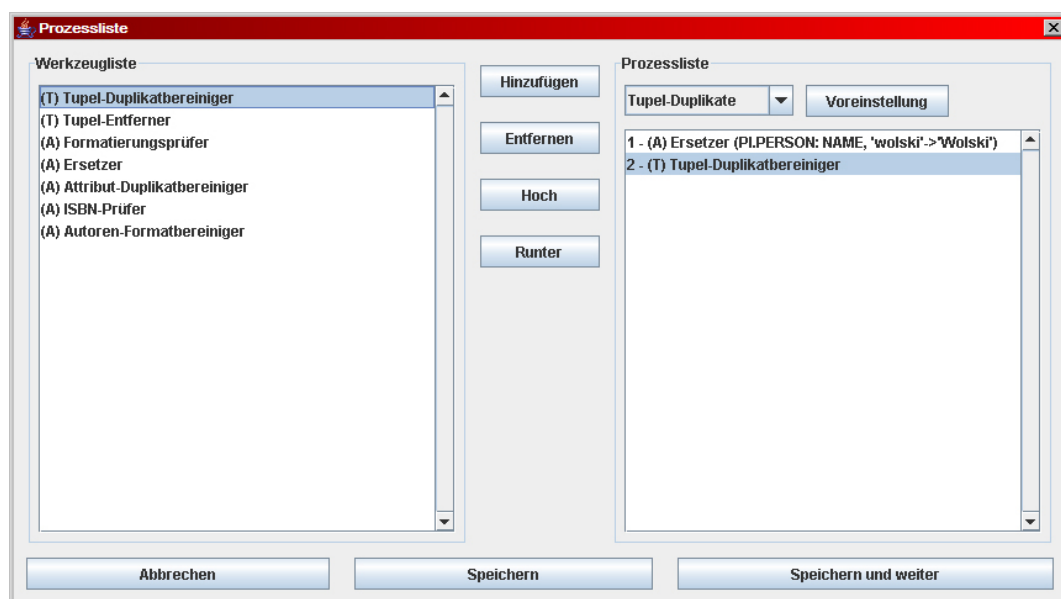


Abbildung 5.8: Prozessliste

Neu hinzugefügte Werkzeuge sind noch unkonfiguriert und müssen im nächsten Schritt eingestellt werden. Die Grundeinstellung eines Werkzeuges ist anhand des Zusatzes im Listeneintrag zu entnehmen.

Mit der Schaltfläche „Speichern“ werden Ihre Änderungen in die Konfiguration übernommen.

Mit der Schaltfläche „Speichern und weiter“ gelangen Sie zur Werkzeugeinstellungen (5.2.5)

Mit der Schaltfläche „Abbrechen“ werden Ihre Änderungen ignoriert und das Fenster geschlossen.

5.2.5 Werkzeugeinstellungen

In diesem Dialog legen Sie die Einstellungen für Ihre Werkzeuge fest. Im linken Panel sehen Sie Ihre Prozessliste. Allen unkonfigurierten Werkzeugen fehlt ein entsprechender Zusatz im Listeneintrag.



Abbildung 5.9: Werkzeugeinstellung: kein gewähltes Werkzeug

Wählen Sie ein Werkzeug, das Sie konfigurieren wollen, und im rechten Panel wird sich der Einstellungsdialog öffnen. Die verfügbaren Optionen variieren je nach Werkzeug. Als Beispiel wird hier ein Tupel-Duplikatbereiniger eingestellt:

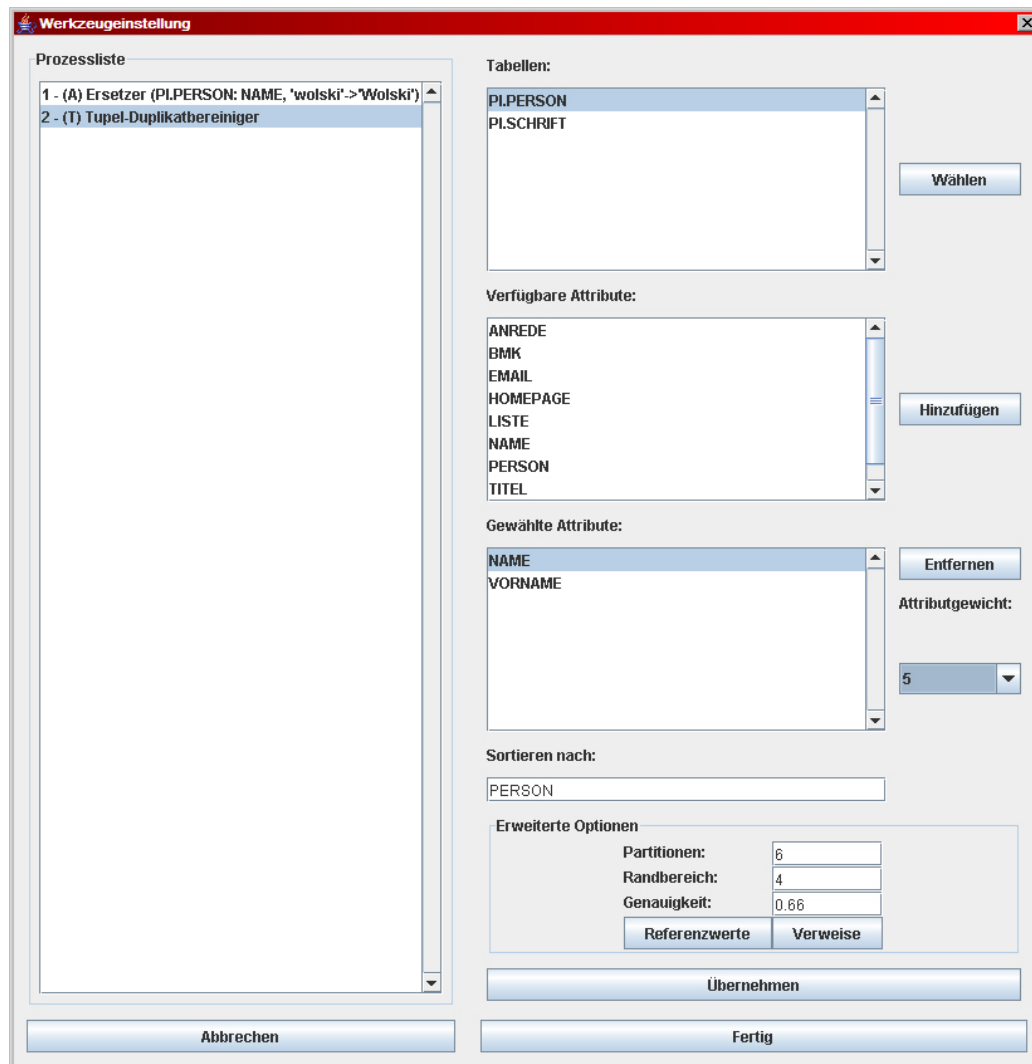


Abbildung 5.10: Werkzeugeinstellung: Tupel-Duplikatbereiniger

Die Abbildung 5.10 zeigt das Panel in voller Ausdehnung. Beachten Sie, dass auf kleineren Bildschirmen Bildlaufleisten angezeigt werden.

Das erste Feld fordert Sie auf, eine der Tabellen aus Ihrer Tabellenkonfiguration (5.2.2) zu wählen. Anschließend wählen Sie die Attribute, anhand derer nach Duplikaten gesucht werden sollen und fügen Sie mit der Schaltfläche „Hinzufügen“ der Attributliste hinzu. Attributen in der Attributliste kann ein Gewicht zwischen 0 und 10 zugeordnet werden. Dieses Gewicht gibt an, wie hoch der Einfluss des Ähnlichkeitsfaktors des Attributes für die Gesamtbewertung ist. Ein Gewicht von 10 ist dabei der stärkste Einfluss.

Das Feld „Sortieren nach“ hat großen Einfluss auf die Genauigkeit des ersten Suchvorganges. Hier wird festgelegt, nach welchem Schlüssel die Tabelle vorsortiert wird. Standardmäßig wird der Primärschlüssel voreingestellt. Meist ist eines der anderen Attribute aber eine bessere Wahl. Sie können auch komplexere Ausdrücke schreiben.

Im Panel mit der Bezeichnung „Erweiterte Optionen“ können Sie die Parameter des Duplikaterkenners modifizieren.

Das Feld „Partitionen“ gibt an, wie stark die Tabelle aufgeteilt wird. Eine hohe Aufteilung beschleunigt den Vergleich innerhalb der Partition. Allerdings vergrößert sich die Anzahl

der Vergleiche mit äußeren Partitionen.

Das Feld „Randbereich“ legt die Anzahl der Randelemente pro Partition. Die Eingabe sollte ein gerader Wert sein. Ein größerer Randbereich verbessert die Genauigkeit des äußeren Vergleichs, die Anzahl der Vergleiche nimmt aber zu.

Mit dem Feld „Genauigkeit“ stellen Sie die Grenze ein, nach der ein Tupel noch als Duplikat erkannt wird.

Die Schaltfläche „Referenzwerte“ öffnet einen Dialog zur Eingabe von Referenzwertvoreinstellungen.

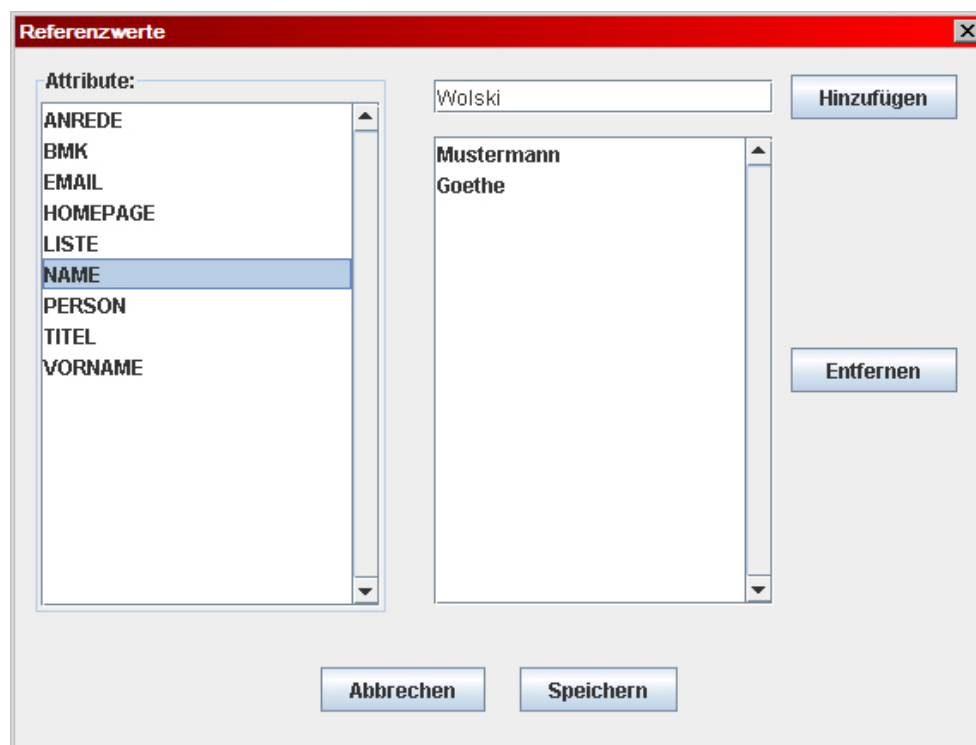


Abbildung 5.11: Werkzeugeinstellung: Referenzwerte

Die Schaltfläche „Verweise“ öffnet einen Dialog zur Konfiguration der Behandlung von durch Foreign-Key referenzierten Tabellen. Standardmäßig werden die Verknüpfungen ignoriert. Dies kann aber dazu führen, dass es bei der Fehlerkorrektur zu einer Foreign-Key-Verletzung kommt.

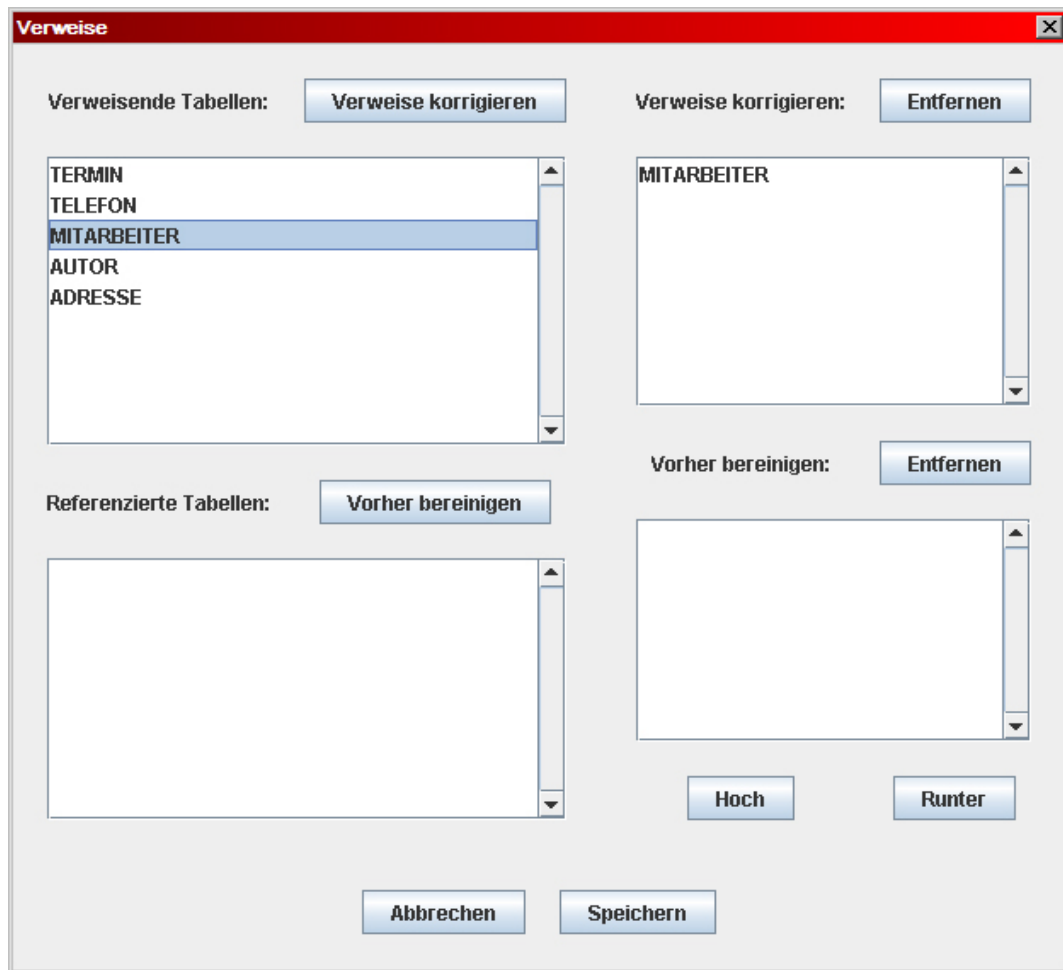


Abbildung 5.12: Werkzeugeinstellung: Verweise

Beachten Sie, dass Ihre Werkzeugeinstellung nur übernommen wird, wenn Sie im rechten Panel auf die Schaltfläche „Übernehmen“ klicken (siehe Abbildung 5.10). Die Einstellung wird sofort aktiv.

Mit der Schaltfläche „Fertig“ schließt das Fenster.

5.3 Hauptoberfläche: Prozess

Auf der Registerkarte „Prozess“ der Hauptoberfläche befinden sich die Steuerelemente um eine Bereinigung zu starten.

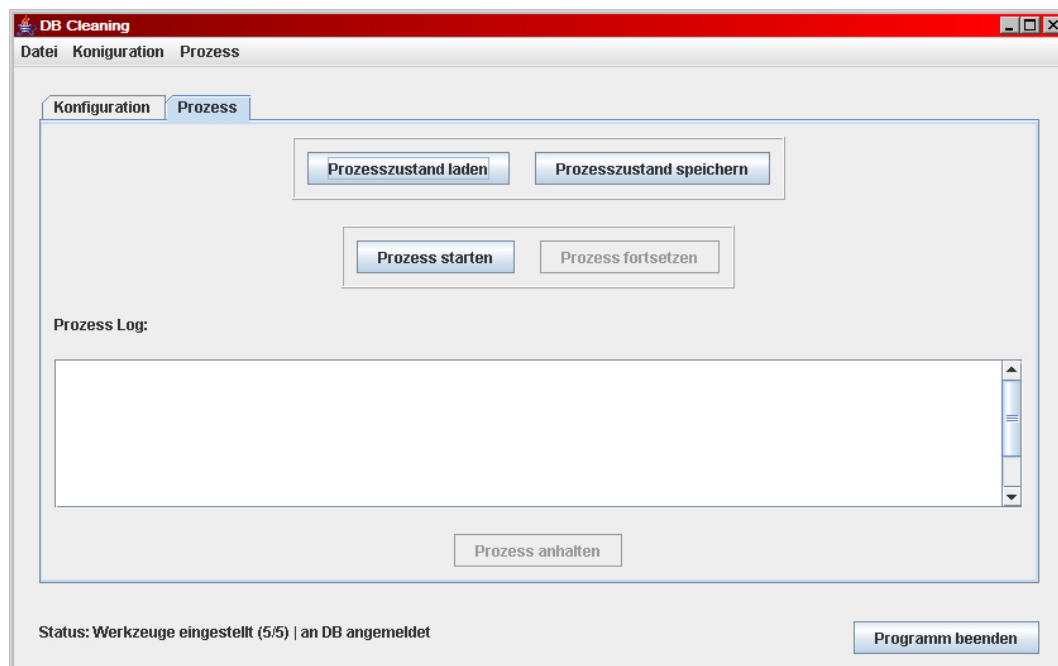


Abbildung 5.13: Hauptoberfläche: Prozess

Das Textfeld in der Mitte der Oberfläche zeigt den derzeitigen Prozesszustand an. Mit den Schaltflächen „Prozesszustand laden“ und „Prozesszustand speichern“ kann der Prozesszustand für eine spätere Fortsetzung archiviert werden. Diese Schaltflächen finden sich auch im Menü „Datei“.

Wenn Sie einen alten Prozesszustand laden wollen, achten Sie darauf, dass Sie zuvor die passende Konfiguration geladen oder erstellt haben, da beim Laden oder Ändern einer Konfiguration der Prozesszustand zurückgesetzt wird. Es ist nicht zu empfehlen, einen Prozesszustand zur falschen Konfiguration fortzusetzen.

Wenn die Konfiguration ausreichend erstellt wurde, wird die Schaltfläche „Prozess starten“ aktiv. Bitte beachten Sie, dass erst nach betätigen der Schaltfläche geprüft wird, ob alle Werkzeugeinstellungen vollständig sind. Das Betätigen dieser Schaltfläche setzt den Prozesszustand zurück und beginnt die Abarbeitung der Prozessliste.

Um einen Prozesszustand fortzusetzen, klicken Sie auf die Schaltfläche „Prozess fortsetzen“. Diese wird nur aktiv, wenn ein angehaltener Prozesszustand geladen ist.

Während ein Prozess läuft, kann die Konfiguration nicht verändert werden. Um einen Prozess anzuhalten, klicken Sie auf die Schaltfläche „Prozess anhalten“. Beachten Sie, dass es etwas dauern kann, bis das Werkzeug reagiert.

Die Schaltflächen dieser Oberfläche finden Sie auch im Menü „Prozess“.

5.4 Interaktion

Wenn ein Werkzeug nicht in der Lage ist, eine Entscheidung automatisch zu treffen, regt es eine Benutzerinteraktion an. Es öffnet sich ein Dialog, dessen Optionen vom Werkzeug abhängig sind. Bei den meisten Werkzeugen wird dies aber der Dialog zur Wahl des Referenzwertes sein.

Merging: Referenzwert

PHW.DBCLEANTEST: PERSON

NAME
Wolski
Wolki

Tupel anzeigen

NAME	VNAME	INS
Wolski	Philipp	DBS

Referenzwert:

Übernehmen

Prozess anhalten **Clusterkorrektur** **Ignorieren und weiter**

Abbildung 5.14: Interaktion: Referenzwerte

Im oberen Panel werden Ihnen die Attributwerte angezeigt, mit denen das Werkzeug derzeit ein Problem hat. Im Dialog zur Wahl des Referenzwertes werden Sie meist zur Wahl eines Wertes zur Ersetzung aller im oberen Panel angezeigten Werte aufgefordert. Im unteren Panel steht Ihnen dazu ein Eingabefeld zur Verfügung.

Um zu einem Wert das gesamte Tupel angezeigt zu bekommen, klicken Sie auf die Schaltfläche „Tupel anzeigen“. Ihnen wird dann das zugehörige Tupel aus der Datenbank angezeigt.

In einigen Fällen können Sie eine Clusterkorrektur fordern. Dazu klicken Sie auf die Schaltfläche „Clusterkorrektur“. Dies ermöglicht es Ihnen, den Cluster zu verändern. Dazu können Sie im obigen Panel nun Tupel an oder abwählen.

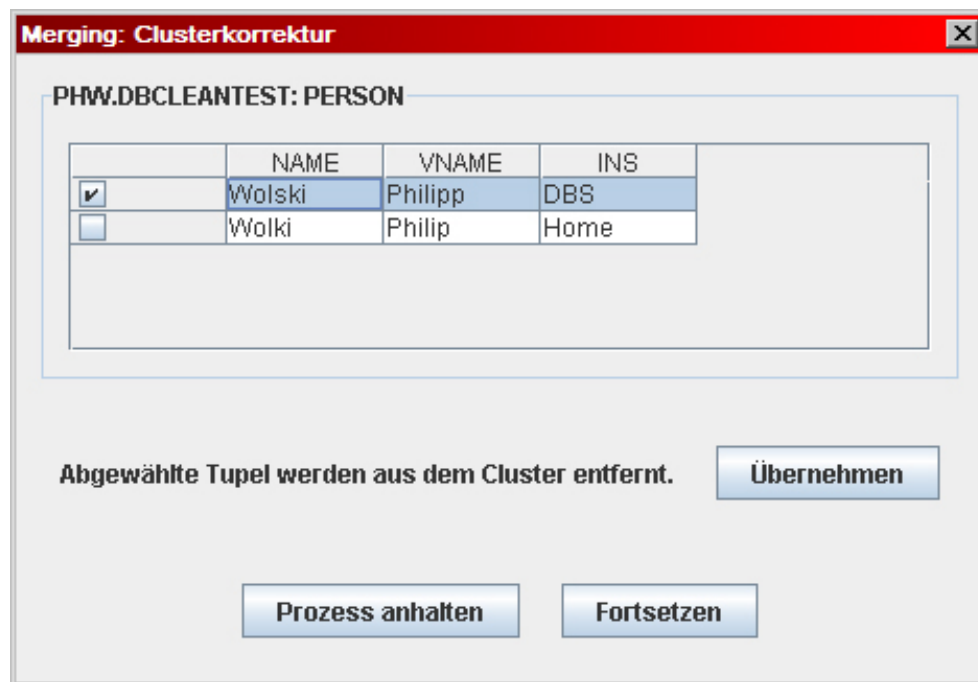


Abbildung 5.15: Interaktion: Clusterkorrektur

Nachdem Sie die Änderung mit der „Übernehmen“ Schaltfläche bestätigt haben, werden die abgewählten Tupel aus dem Cluster entfernt.

Der Prozessablauf wird solange angehalten, bis Sie die Interaktion beendet haben. Dies dient der Datensicherheit für nachfolgende Werkzeuge. Während der Interaktion kann der Prozess auch angehalten werden.

Kapitel 6

Testbericht

Wir wollen hier eine Bereinigung der Schlagworte versuchen. Hierzu kommt der Attribut-Duplikatbereiniger zum Einsatz. Dieses Beispiel ist leicht zu konfigurieren und einfach nachzuvollziehen. Dennoch wird hier die Erkennungsrate von PC-Filter und unserem modifizierten Field-Similarity Verfahren scharf geprüft, wie sich später zeigen wird. Aus Sicherheitsgründen wird eine Kopie der Schlagworttabelle genutzt.

6.1 Vorbereitung

Wir stellen die Frameworkeinstellungen wie in Abschnitt 5.2 ein. In der Tabellenkonfiguration wählen wir jedoch die kopierte Tabelle Schlagworte.

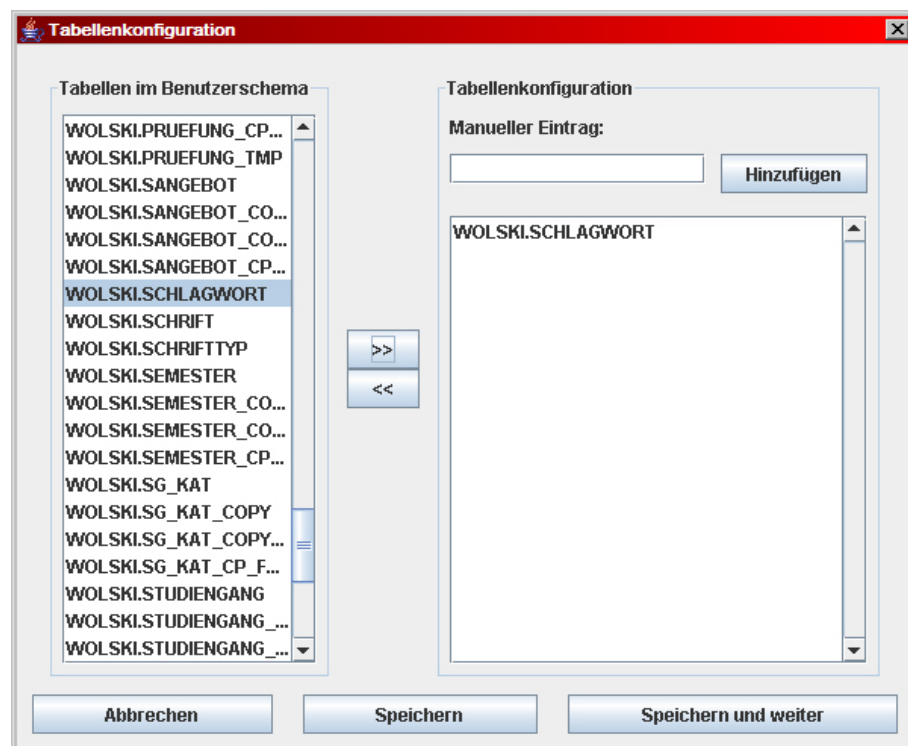


Abbildung 6.1: Schlagworttabelle wählen

Wir stellen bei der Attributkonfiguration sicher, dass für das Attribut „*SCHLAGWORT*“ **VARCHAR2** gewählt ist. Anschließend stellen wir das Werkzeug ein. Als Genauigkeit wählen wir für den ersten Versuch 0,75. Als Attribut wählen wir nun „*SCHLAGWORT*“ aus.

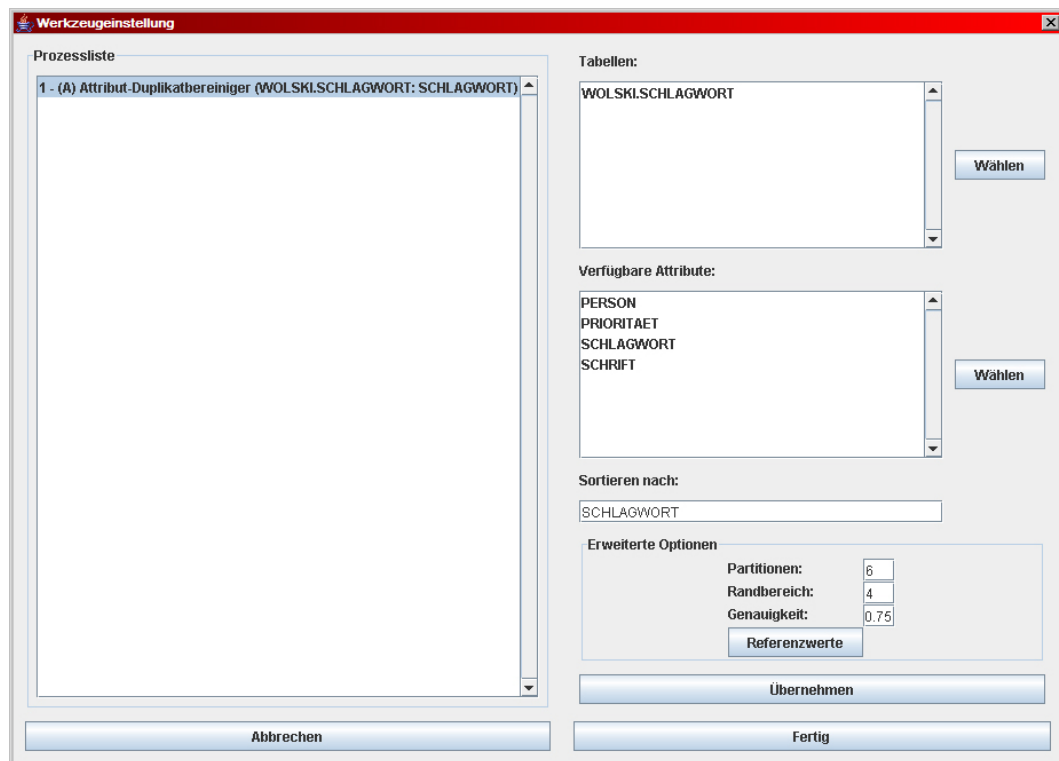


Abbildung 6.2: Attribut-Duplikatbereiniger einstellen

Mit diesen Einstellungen können wir nun einen Prozess starten. In dem Prozesslog können wir verfolgen, wie die Blöcke von PC-Filter durchlaufen werden.

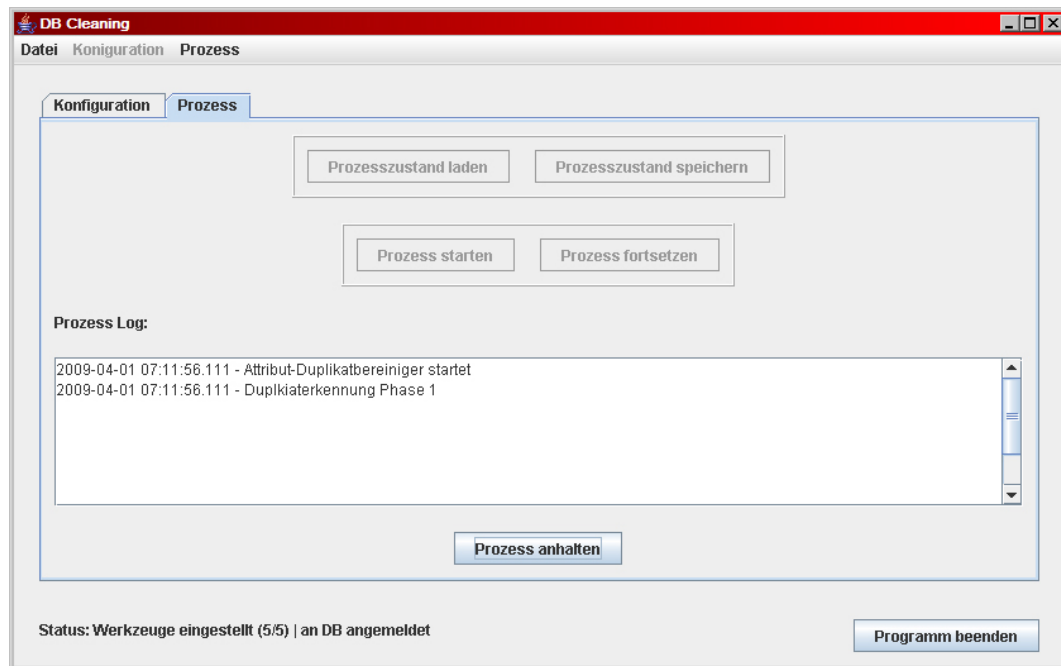


Abbildung 6.3: Laufender Prozess

6.2 Laufzeittest

Die Schlagworttabelle hat 29922 Einträge. Erprobt wurden die Verfahren auf einer Tabelle mit 313 Einträgen. Zusätzlich kommen in der Schlagworttabelle Attributwerte vor, die aus mehreren Wörtern bestehen.

Folgende Zeiten wurden für die Ausführung mit PC-Filter gemessen:

- Alle sechs Prozeduren auf Testtabelle nur mit Field-Similarity: 1,5 Sekunden.
- Testtabelle mit der modifizierten Ähnlichkeitsfunktion: 5 Sekunden.
- Schlagworttabelle erste Prozedur: 2 Sekunden.
- Schlagworttabelle zweite Prozedur: 285 Sekunden.
- Schlagworttabelle dritte Prozedur: 0,5 Sekunden.
- Schlagworttabelle vierte Prozedur: 0,2 Sekunden.
- Schlagworttabelle fünfte Prozedur: > 25 Minuten.
- Schlagworttabelle sechste Prozedur: Messung abgebrochen.

Das manuelle Anwenden des Mergingverfahrens auf die unvollständige Duplikattabelle ergibt nahezu keine Interaktionsanfragen. Eine Anfrage an die Tabelle zeigt, dass sich in der Schlagworttabelle sehr viele Attributwerte hoher Ähnlichkeit befinden. Der abgebrochene Vorgang hat bereits 2815 Duplikate gefunden. Bei einer hohen Zahl an Duplikaten werden die Verfahren deutlich langsamer. Zu erklären ist dies daran, dass Vergleiche mit äußeren

Partitionen viel eher als Abhängigkeit erkannt werden als bei eher diversem Datenbestand. Da die Verfahren keine lernenden Verfahren sind, wird das Wiederkehren bereits als Duplikate erkannter Werte ignoriert und eine neue Prüfung angestoßen. Tabellen mit vielen Duplikaten führen ebenfalls dazu, dass das Merging mehr Zeit braucht, um alle als Cluster zusammenhängenden Tupel zu finden. Fehlerkennungen bleiben bei vielen Duplikaten auch häufiger unentdeckt. Um dennoch Schreibfehler finden zu können, müsste die Duplikattabelle auf eher unähnliche Tupel geprüft werden. Eine einfache Lösung wäre, die Vorauswahl die PC-Filter trifft einzuschränken. So kann mit einem „Distinct“ vor der Cursoröffnung ein Großteil der überflüssigen Duplikate vermieden werden. Dies wäre ein Ansatz, um die Werkzeugsuite noch zu erweitern.

Aus den bereits erkannten Duplikaten geht ebenso noch hervor, dass die gewählte Genauigkeit, bei der hohen Ähnlichkeit zueinander, zu gering war.

1273	default reasoning - default inheritance
1274	default reasoning - default inheritance
1275	default reasoning - default inheritance
1276	default reasoning - default inheritance
1277	default reasoning - default inheritance
1278	default reasoning - default inheritance
1279	default reasoning - default inheritance
1280	default reasoning - default priorities
1281	default reasoning - default priorities
1282	default reasoning - default priorities
1283	default reasoning - default priorities
1284	default reasoning - default priorities
1285	default reasoning - default priorities
1286	default reasoning - default priorities
1287	default reasoning - default priorities
1288	default reasoning - default priorities
1289	default reasoning - default reasoning
1290	default reasoning - default reasoning
1291	default reasoning - default reasoning
1292	default reasoning - default reasoning
1293	default reasoning - default reasoning
1294	default reasoning - default reasoning
1295	default reasoning - default reasoning
1296	default reasoning - default reasoning
1297	default reasoning - default reasoning
1298	default reasoning - default reasoning
1299	default reasoning - default reasoning
1300	default reasoning - default reasoning
1301	default reasoning - default reasoning

Abbildung 6.4: Zu viele ähnliche Werte

Kapitel 7

Implementierung

Hier wird die Implementierung der unter Kapitel 4 vorgestellten Werkzeugsuite erläutert. Zu den Methoden der wichtigsten Klassen finden sich in der Schnittstellen-API im Anhang A nähere Erläuterungen.

7.1 Umgebung

Die Anwendung ist für die Java Versin 1.5.0_17 implementiert und nutzt den Oracle JDBC Treiber in der Version 10.2.0.4. Zur Verbindung mit der Datenbank müssen die externen Bibliotheken des OCI Instant Clients in der Version 10.2.0.4 der Anwendung zugänglich gemacht werden. Dazu müssen diese sich im Klassenpfad der Anwendung befinden. Zum Ablegen von Konfigurationsdaten in eine externe Datenstruktur kommt die JDOM Bibliothek in der Version 1.1 zum Einsatz. Diese erlaubt das Anlegen und Einlesen von Dateien im *XML* Format.

7.2 Entwurf und Aufbau

Der Entwurf der Java-Anwendung folgt den Leitlinien aus dem Kapitel 4. Die Aufteilung der Klassen soll, soweit es sinnvoll angewendet werden kann, dem MVC-Muster folgen. Dieser Abschnitt soll einen Überblick über die Realisierung der Anwendungsstruktur als Java-Anwendung geben.

7.2.1 Paketübersicht

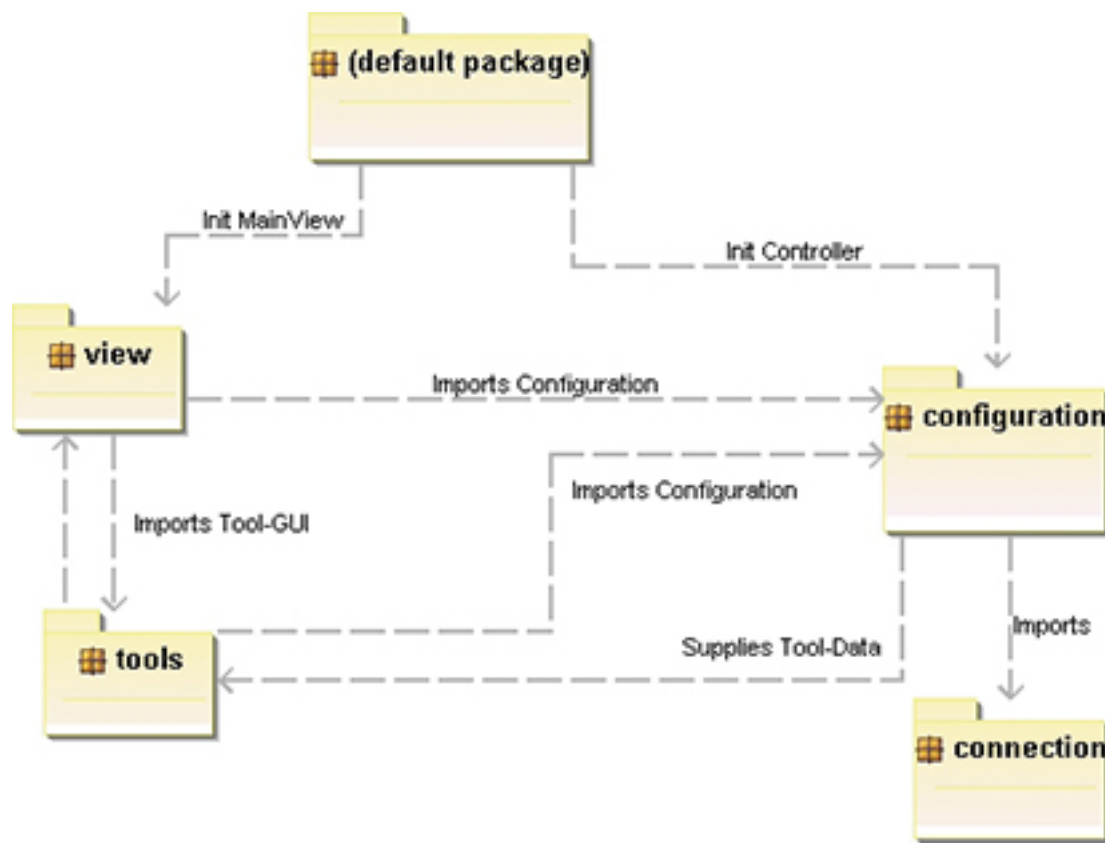


Abbildung 7.1: Paket-Diagramm

Das **Paket (default package)** enthält den Programmeinstiegspunkt, also die Startklasse. Von hier werden die Konfiguration und die Oberfläche initialisiert.

Das **Paket View** beinhaltet die Klassen zur Darstellung der Bedienoberfläche und stellt also die „View“ dar.

Das **Paket Configuration** beinhaltet die „Controller“-Strukturen und stellt den Konfigurations- und Initialisierungsblock dar. Es verwaltet die internen Speicherstrukturen und führt die Konfigurations- und Prozessliste. Die Werkzeuge registrieren sich an der Konfiguration und sind dann im Projekt für andere Pakete sichtbar. Als Hilfsprozeduren stellt es die Kommunikation mit der Datenbank bereit. Das Datenbankverbindungsobjekt wird dabei mit der Konfiguration zusammen initialisiert und durch die gespeicherten Verbindungsdaten konfiguriert.

Das **Paket Connection** ist ein Unterstützungspaket für den „Controller“ und beinhaltet Klassen zur Datenbankkommunikation, wie das Datenbankverbindungsobjekt.

Das **Paket Tools** beinhaltet die Werkzeugklassen. Dies umfasst die Funktionsblöcke der Benutzerebene und, abhängig vom Werkzeug, die Vorbereitung des „Model“-Teils des MVC-Musters. Die Klassen des *Pakets Tools* behandeln ausschließlich die werkzeugrelevanten Operationen. Kommunikation mit den Werkzeugen erfolgt über dieses Paket. Damit Werkzeuge dem Projekt bekannt gemacht werden, müssen diese sich an der Konfiguration angemeldet haben. Die Datenbankabfragen werden über die, aus dem *Paket*

Configuration importierten Datenbankobjekte, abgewickelt. Für die Anzeige der werkzeugrelevanten Oberflächen stellt es dem *Paket View* die notwendigen Informationen zur Verfügung.

7.2.2 Klassenübersicht

Dieser Abschnitt soll eine Übersicht über die, für den Programmaufbau relevanten Klassen geben und diese kommentieren.

7.2.2.1 Startklassen

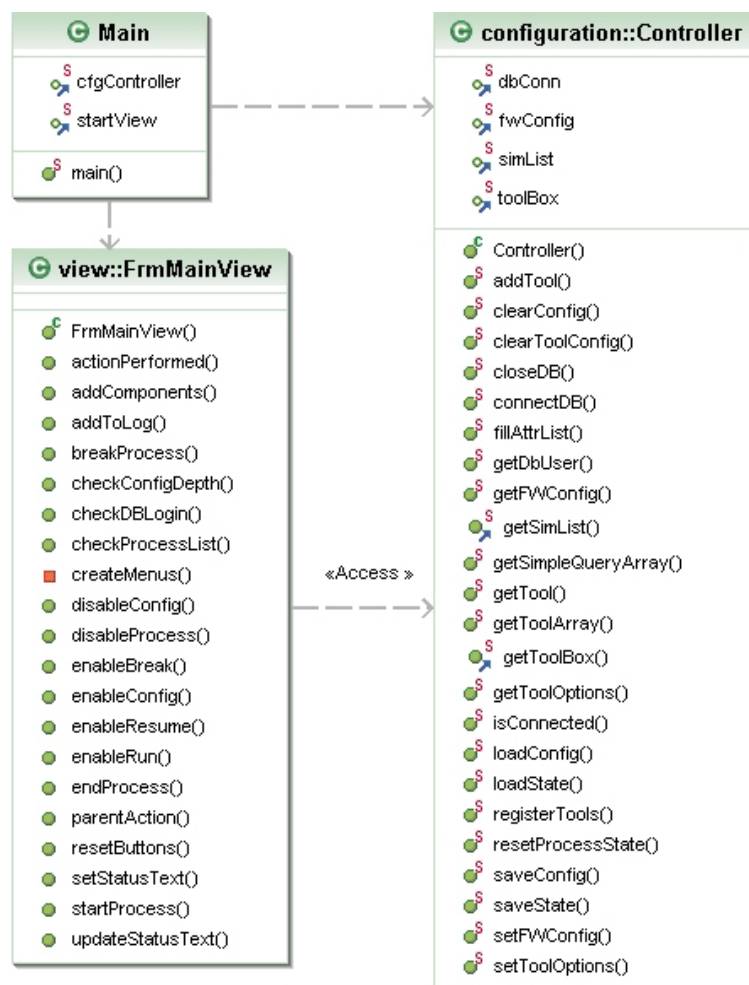


Abbildung 7.2: Klassendiagramm Hauptoberfläche

Die **Klasse Main** initialisiert die konfigurationsverwaltende *Klasse Controller* aus dem *Paket Configuration* und die Hauptoberfläche aus dem *Paket View*. Sie ist relevant als Programmeinstiegspunkt.

Die **Klasse Controller** ist die zentrale Konfigurationsverwaltung. Die Klasse verwaltet die Frameworkkonfiguration, das Datenbankobjekt und verfügt über Prozeduraufrufe, um die Konfiguration als externe Datei abzulegen. Durch den Namen der Klasse ist bereits naheliegend, dass diese Klasse das Hauptelement des „*Controller*“-Parts ist. Sie stellt aber

nicht den gesamten „*Controller*“ dar und sollte daher im Folgenden nicht verwechselt werden. Diese Klasse ist als statische Klasse implementiert. Dadurch können deren Methoden von jeder Klasse ohne Instanzierung aufgerufen werden. Dahingegen instanziiert die Klasse die Konfiguration und Datenverbindung.

Die **Klasse FrmMainView** stellt die Hauptoberfläche dar. Dialogfenster überlagern jeweils diese Hauptoberfläche. Sie dient damit neuen Fenstern als „Parent“-Klasse. Die Konfiguration und der Prozesszustand haben unmittelbare Auswirkung auf deren Oberflächenverhalten. Sie kommuniziert dabei mit der *Klasse Controller* und stößt notwendige Funktionsblöcke durch Auswerten der Benutzerinteraktion an.

7.2.2.2 Paket View

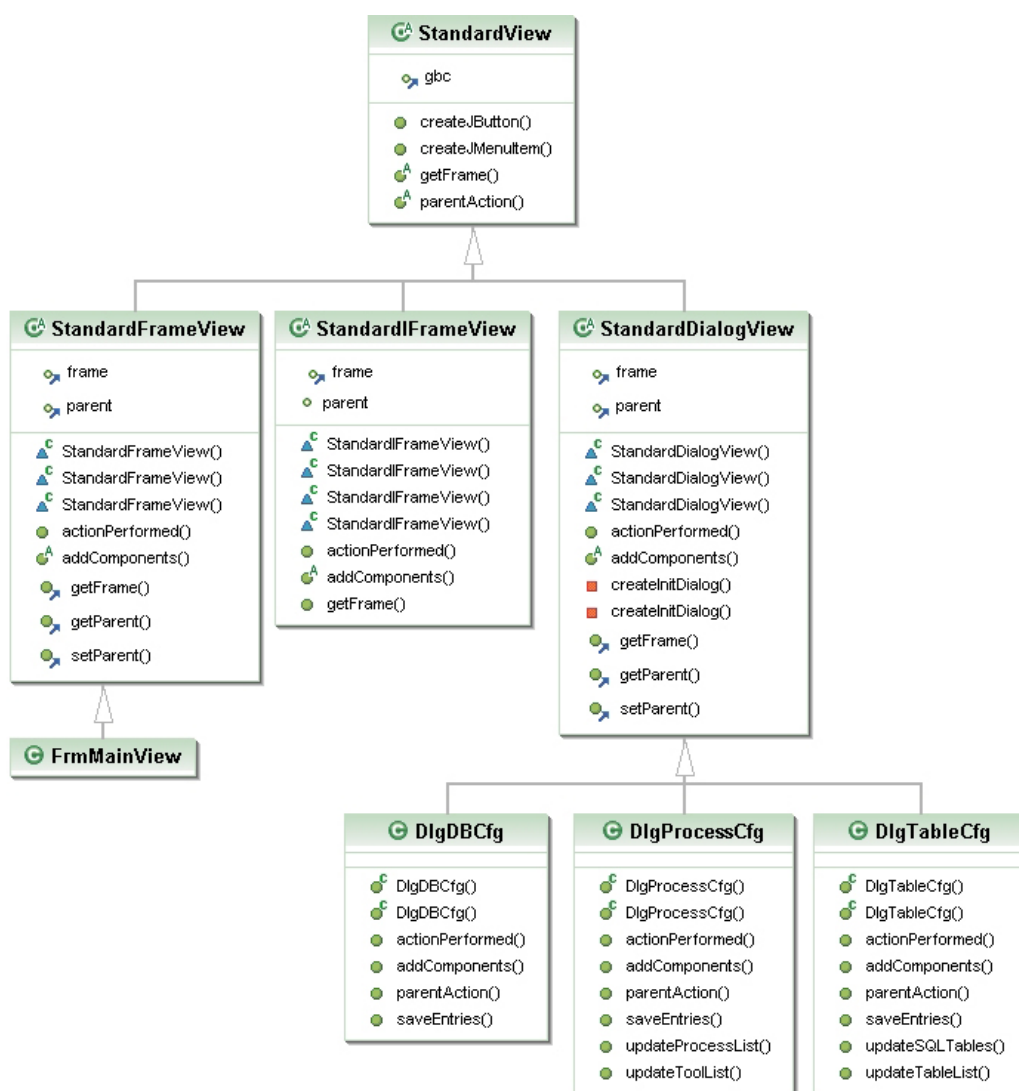


Abbildung 7.3: Klassendiagramm 1 von Paket View

Aus Gründen der Übersichtlichkeit wurde auf die Vererbungsfeile in den Abbildung 7.3 und 7.4 verzichtet.

Die **abstrakte Klasse StandardView** stellt die generische Instanz einer „*View*“-Klasse dar. Sie implementiert die Methode `parentAction()`, durch die neu generierte Fenster

dem aufrufenden Fenster Nachrichten zukommen lassen kann. Die *StandardView* Klassen dienen als Unterstützerklassen, um den Fensteraufbau zu vereinheitlichen und vorimplementieren den notwendigen „Controller“-Part für die Interaktionselemente. Diese Klassen vereinheitlichen also den Aufbau der steuernden Klasse, während die später genannten *InitView* Klassen den Fensteraufbau vereinheitlichen.

Die **abstrakte Klasse StandardFrameView** ist die Klasse, von der Frame-Fenster abgeleitet werden. Die Klasse übersetzt den Datentyp JFrame in die generische Form. Sie erstellt im Konstruktor einen *InitFrame*.

Die **abstrakte Klasse StandardIFrameView** ist die Klasse, von der InternalFrame-Fenster abgeleitet werden. Die Klasse übersetzt den Datentyp JInternalFrame in die generische Form. Sie erstellt im Konstruktor einen *InitIFrame*.

Die **abstrakte Klasse StandardDialogView** ist die Klasse, von der Dialog-Fenster abgeleitet werden. Die Klasse übersetzt den Datentyp JDialog in die generische Form. Sie erstellt im Konstruktor einen *InitDialog*.

Die **Klassen DlgProcessCfg, DlgTableCfg, DlgDBCfg** sind drei ausgewählte Klassen zur Demonstration der Implementierung von Konfigurationsdialogen. Abgeleitet von StandardDialogView und mit dem Fenster vom Typ InitDialog, lassen sich diese Klassen über die generische Instanz beschreiben. Damit können sowohl beliebige Klassen diese Klassen instanzieren als auch beliebige weitere Fensterklassen über die Dialoge instanziiert werden.

Die Methode *saveEntries()* stellt dabei die Übernahmefunktion der Einstellungen in die Konfiguration dar. Der Prozeduraufruf ist nicht vererbt, sondern stellt die Verbindung zwischen der vom „Controller“-Element verwalteten Konfiguration und der Bedienoberfläche der „View“-Elemente dar (vgl. Abbildung 4.2).

Die Methode *parentAction()* wurde aus der Ableitung von der **abstrakte Klasse StandardView** implementiert. Mit dieser kommunizieren Fenster mit deren aufrufenden Fenstern. Insbesondere die erwähnten Dialogfenster geben der Klasse *FrmMainView* deren Zustand zurück, der dort weiter ausgewertet wird. So weiß am Ende der „Controller“ immer, in welchem Prozesszustand sich die Anwendung befindet, da auch Interaktionsfenster diesem Schema folgen.

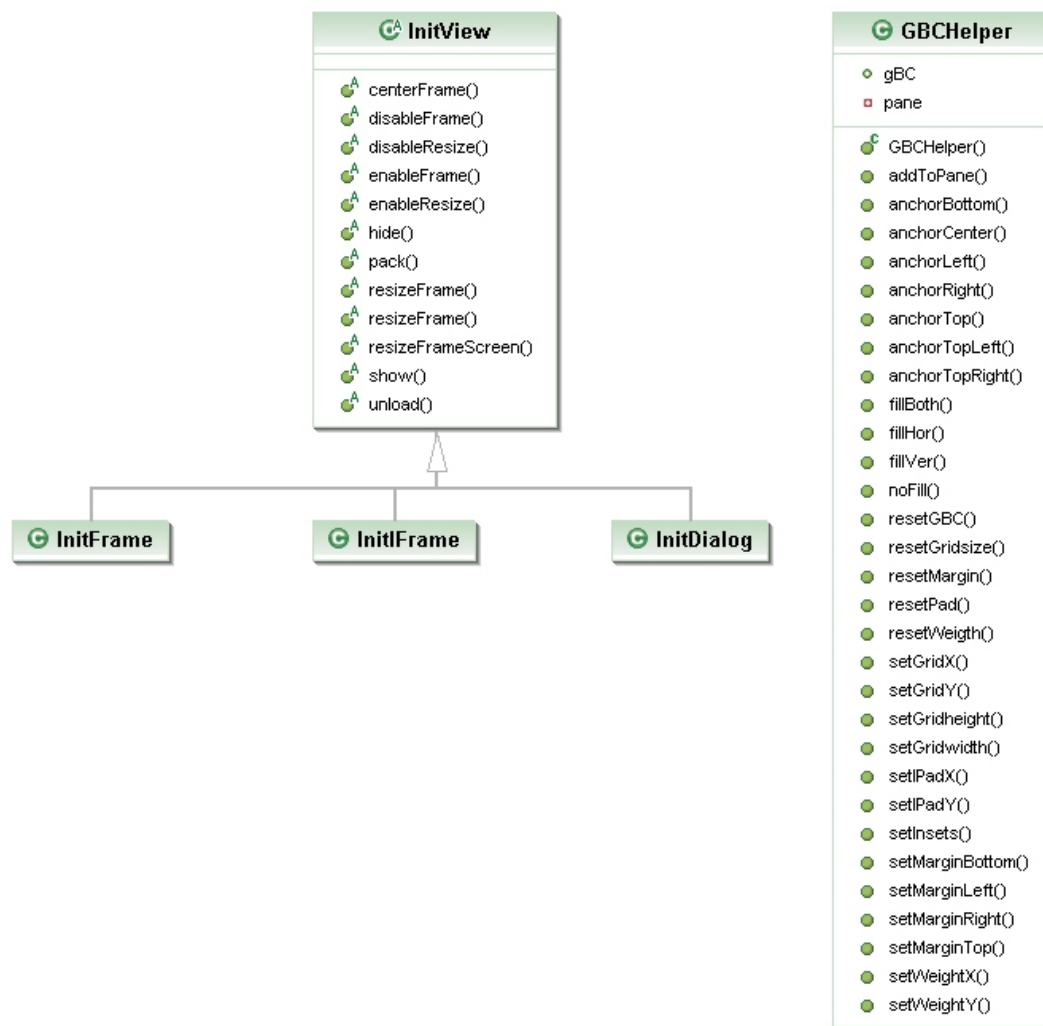


Abbildung 7.4: Klassendiagramm 2 von Paket View

Die **abstrakte Klasse InitView** ist die generische Instanz eines Fensters im *Paket View*. Sie implementiert Prozeduren, durch die komplexe Fensterkonfigurationen vereinfacht werden. Dies ermöglicht, durch die Ableitung von der generischen Instanz, die Konfiguration von Fenstern unabhängig von deren zugrunde liegenden Datentyp.

Die **Klasse InitFrame** erzeugt ein JFrame-Fenster.

Die **Klasse InitIframe** erzeugt ein JInternalFrame-Fenster.

Die **Klasse InitDialog** erzeugt ein JDialog-Fenster.

Die **Klasse GBCHelper** erzeugt ein GridBagLayout und vereinfacht das Positionieren der Elemente durch zusätzliche Hilfsprozeduren. Neben dem Grundgedanken, das GridBagLayout vereinfacht konstruieren zu können, ist es mit dieser Klasse möglich, die Layoutgenerierung zu vereinheitlichen.

7.2.2.3 Paket Configuration

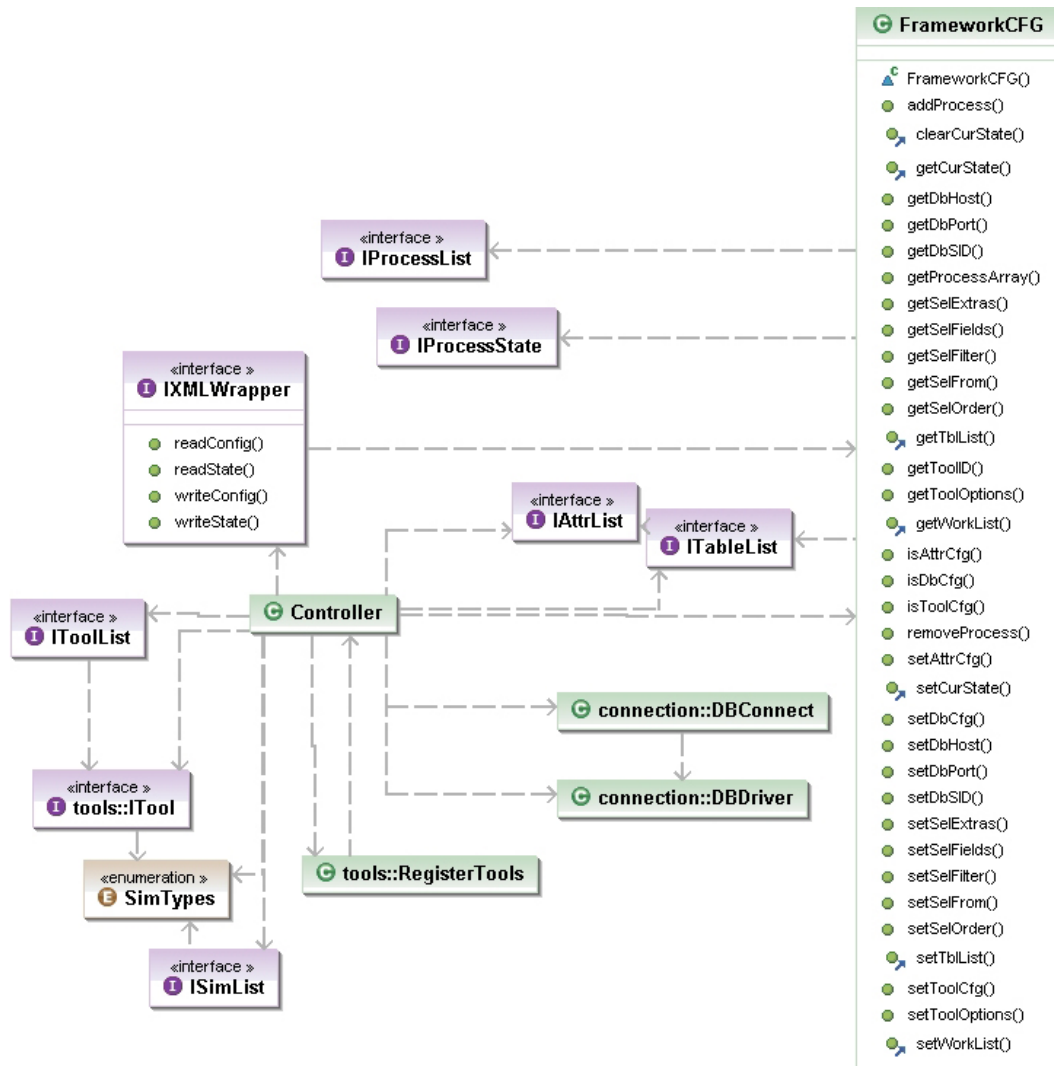


Abbildung 7.5: Klassendiagramm Paket Configuration

Die meisten, der in Abbildung 7.5 gezeigten Klassen, bilden interne Datenstrukturen. Der genaue Aufbau dieser Strukturen wurde aus Gründen der Übersichtlichkeit weggelassen.

Über die **Klasse RegisterTools** des *Paket Tools* werden die im Projekt verfügbaren Werkzeuge der Konfiguration bekannt gemacht, indem diese in einer von der *Klasse Controller* geführten Liste vom Typ *IToolList* eingetragen werden. Diese Liste enthält Elemente des Typs *ITool*, der im Abschnitt 7.2.3 näher beschrieben wird.

Die **Klasse FrameworkCFG** verwaltet die Konfiguration und Prozessliste und wird von der Klasse *Controller* instanziiert. Die Klasse stellt damit die Repräsentation der Datenquelle für die Konfiguration dar. Diese wird dabei in lokalen Datenstrukturen gespeichert, die über Interface instanziiert werden. So ist die Prozessliste vom Typ *IProcessList*, und der Prozesszustand wird in einer Datenstruktur des Typs *IProcessState* festgehalten. Zum Ablegen der Konfiguration in eine externe Datei muss die *Klasse Controller* folglich alle in der *Klasse FrameworkCFG* zentralisierten Werte ablegen. Dazu wird mit dem Interface *IXMLWrapper* eine Implementation eines XML Schreibers, bzw. Lesers von der *Klasse*

Controller instanziiert, dessen Methoden die, in der Klasse *FrameworkCFG* abgelegten Datenstrukturen, verarbeiten können. Der eigentliche Datenzugriff auf gespeicherte Daten übernimmt dabei die Bibliothek JDOM.

7.2.2.4 Paket Tools

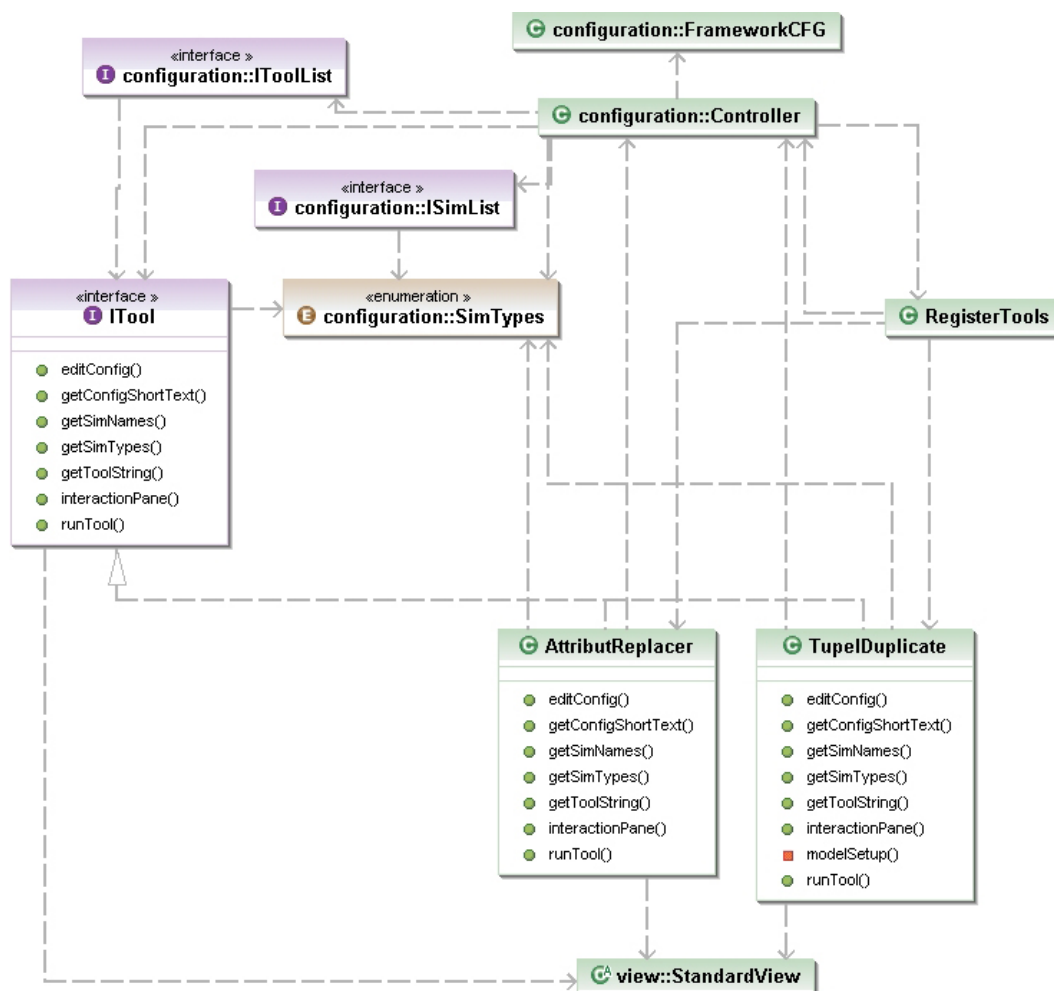


Abbildung 7.6: Klassendiagramm Paket Tools

In der Abbildung 7.6 sind die Zusammenhänge der Klassen des **Paket Tools** mit denen der verbundenen anderen Pakete dargestellt. Aus Gründen der Übersichtlichkeit reduzieren wir uns auf die Klassen **Klasse TupelDuplicate** und **Klasse AttributReplacer**. Diese stellen jeweils ein kompliziertes und ein einfaches Werkzeug dar. Beide Klassen implementieren das **Interface ITool** und werden durch die **Klasse RegisterTools** der *Klasse Controller* übergeben. Es ist ersichtlich, dass beide Werkzeuge einen durch das *Interface ITool* vererbten Aufbau haben. Dies hat den Grund, dass jedes Werkzeug von den aufrufenden Klassen gleich behandelt werden soll. Das heißt, die Kommunikation mit dem Werkzeug verläuft über die, durch das Interface vererbten Methoden. Damit ist es möglich, beliebige Werkzeuge zu realisieren und diese mit nur einem Eintrag in die *Klasse RegisterTools* dem Projekt zu Verfügung zu stellen. Dazu sind folgende Methoden von besonderer Bedeutung:

Die Methode ***editConfig()*** erzeugt ein JPanel, dass die Oberfläche für die Werkzeugeinstellungen darstellt.

Die Methode ***runTool()*** beinhaltet den Ausführungsblock des Werkzeuges. Hier werden nacheinander die einzelnen Schritte ausgeführt und auf Unterbrechungen reagiert.

Die Methode ***interactionPane()*** erzeugt ein JPanel, dass die Oberfläche für die Benutzerinteraktion darstellt.

Die Klasse *TupelDuplicate* besitzt noch die private Methode ***modelSetup()***. Dabei handelt es sich um eine Methode, die SQL Anweisungen zum Generieren der PL/SQL Prozeduren sammelt, um die *runTool()* Methode übersichtlich zu behalten.

7.2.3 Aufbau von Werkzeugen

Werkzeuge werden als Klassen im *Paket Tools* implementiert. Dabei muss eine solche Klasse mindestens folgendes leisten:

- Es muss eine Prozedur zur Beschreibung eines Konfigurationsdialogs für die „View“ existieren.
- Es muss eine Prozedur zur Beschreibung eines Interaktionsdialogs für die „View“ existieren.
- Es muss eine Prozedur zum Ausführen des Werkzeuges geben.
- Die Klasse muss mit dem „*Model*“-Part kommunizieren.
- Das Werkzeug muss auf Abbruchkommandos reagieren.

Da die Methoden, die diese Vorgaben erfüllen, die Grundlagen für den Zugriff über die Klassen des *Paket View* und *Paket Configuration* bilden, sind diese im **Interface ITool** definiert. Ein Werkzeug implementiert dieses Interface. So kann jedes Werkzeug unabhängig von seiner genauen Funktion konfiguriert und ausgeführt werden.

Werkzeuge werden durch die Klasse *RegisterTools* an der Konfiguration registriert. Das Werkzeug bekommt eine Identifikation und wird als instanziiertes Objekt der Liste vom Typ *IToolList* hinzugefügt. Von dort können die Werkzeuge in die Prozessliste aufgenommen werden. Jedes Element der Prozessliste stellt einen eigenen Prozess dar. Es werden zu jedem Prozess die Werkzeugeinstellungen und einen Verweis auf das jeweilige Werkzeug abgelegt. (Vgl. Klasse *FrameworkCFG* in Abschnitt 7.2.2.3). Dadurch genügt die einmalige Instanzierung eines Werkzeuges durch die Klasse *RegisterTools*.

Bisher nicht beachtet wurde das in Abbildung 7.6 und 7.5 vorkommenden **Enum SimTypes** sowie das **Interface ISimList**. Da jedes Werkzeug ein abgeschlossenes System darstellt, ist es der Anwendung nicht bekannt, welche optimierten Verfahren zur Ähnlichkeitsfaktor erzeugung für bestimmte Domämentypen vom Werkzeug unterstützt werden. Beim Anmelden des Werkzeuges wird nun die Information vom Werkzeug abgerufen, welche Domämentypen es unterstützt und diese in die Liste vom Typ *ISimList* eingetragen. Da nun nicht jedes Werkzeug die gleichen Domämentypen kennt, wird zusätzlich zu dem Domämentyp noch der Datentyp, auf dem der Domämentyp basiert, in die Liste aufgenommen. Damit ist es anderen Werkzeugen möglich, bei unbekanntem Domämentyp das

Ähnlichkeitsmaß über den Datentyp zu wählen. Die unterstützten Standarddatentypen befinden sich in dem zuvor genannten *Enum SimTypes*.

Bei der Abarbeitung der Prozessliste wird die Prozedur zum Ausführen des Werkzeuges gestartet. Während sich das Werkzeug in der Ausführung befindet, muss es auf Abbruchkommandos reagieren können. Da mit der Abarbeitung der Prozessliste das Programm in den Prozess, bzw. Werkzeugblock wechselt, würde die Auswertung der Bedienoberfläche zurückgestellt bis der Block vollendet wurde. Daraus ergibt sich die Notwendigkeit, den Prozessblock als eigenen Thread zu starten und durch globale Signale mit der Bedienoberfläche zu kommunizieren.

Während der Abarbeitung wird der aktuelle Block und alle notwendigen Ausführungsinformationen im Prozesszustand geführt. Dieser wird vom Werkzeug der Klasse *FrameworkCFG* mitgeteilt.

Der „*Model*“-Part der Werkzeuge kann temporäre Tabellen und PL/SQL Prozeduren umfassen, die vor der Ausführung nicht auf der Datenbank vorhanden sind. Eine Werkzeugklasse hat folglich die Aufgabe dafür zu sorgen, dass auf der Datenbank die benötigten Werkzeugelemente vorhanden sind. Grob gefasst werden beim Ausführen eines Werkzeuges fehlende Werkzeugelemente auf der Verarbeitungsebene generiert. Dies kann zum Beispiel durch das Erstellen von PL/SQL Prozeduren auf der Datenbank geschehen. Dies kann der Fall sein, wenn ein Verfahren, zum Beispiel aus Abschnitt 7.5, gebraucht wird. Ein solches Verfahren lässt sich mit einer statischen Query auf der Datenbank erstellen und auch wieder entfernen.

Nach Abarbeitung eines Werkzeuges sollten die auf der Datenbank neu erstellten temporären Elemente wieder entfernt werden, um unnötige Datenlast zu vermeiden.

7.3 Implementation vom Prozesszustand

Hier soll kurz auf den Zusammenhang zwischen dem Prozesszustand, der Prozessliste, einer Interaktion und der Unterbrechung eingegangen werden. Eine Interaktion stellt nach 4.3 eine „Exception“ dar. Die Natur einer „Exception“ ist das Unterbrechen eines Ablaufs. Nun wird eine Interaktionsanfrage nicht zufällig, sondern an genau definierten Stellen generiert. Nämlich dann, wenn der Prozess angehalten werden kann, ohne dass dadurch Daten verloren gehen. Die Interaktion verhält sich in diesem Zusammenhang wie eine Unterbrechung. Aus dieser Erkenntnis können wir ableiten, dass sowohl die Unterbrechung als auch die Interaktion den Prozess durch die Informationen des Prozesszustandes fortsetzt.

Ein Werkzeug wird allgemein durch die *runTool()* Methode gestartet. In dieser Methode wird ebenfalls entschieden, die Ausführung anzuhalten. Daraus resultiert, dass diese Methode, anhand des Prozesszustandes, den Einstiegspunkt in die Ausführung des Werkzeuges bestimmt.

Die Prozessliste sei implementiert als Liste von Prozessen. Prozesse haben eine Identifikation. In einem Prozess wird ein Werkzeug ausgeführt. Ein Werkzeug hat ebenfalls eine Identifikation. Wir können folgern, dass in der Prozessliste eine Prozess-Id auf eine Werkzeug-Id verweisen sollte. Des Weiteren haben Werkzeuge spezifische Einstellungen, die sich zwischen Prozessen unterscheiden. Die Werkzeugeinstellungen gehören nicht dem

Werkzeug sondern dem Prozess. Die Prozessliste ergibt sich nun aus der Zuordnung einer Prozess-Id zu einem Werkzeug und dessen Einstellungen.

Hieraus folgert, dass die Methode, die den Prozessblock startet, anhand der, im Prozesszustand abgelegten Prozess-Id, aus der Prozessliste das passende Werkzeug startet. Das gestartete Werkzeug wird anschließend aus dem Prozesszustand seinen Einstiegspunkt finden und gemäß den Werkzeugeinstellungen der Prozessliste fortfahren. Es zeigt sich, dass der Prozesszustand sowohl aus der Prozess-Id der Prozessliste besteht, als auch aus Daten, die frei vom Werkzeug abgelegt werden können.

7.4 Implementation von Werkzeugen

Im Abschnitt 7.2.2.4 wurden bereits zwei Klassen vorgestellt, die sich mit der Implementation von Werkzeugen beschäftigen. Dabei steht die *Klasse* *AttributReplacer* für das einfache Bereinigen wertunabhängiger Fehler, und deren Aufbau kann analog auf alle in Abschnitt 4.4 erwähnten wertunabhängig agierenden Werkzeuge übertragen werden. Die *Klasse* *TupelDuplicate* stellt die Implementation des Tupel-Duplikatbereinigers aus 4.4.1.1 dar, und umfasst die Realisierung von Unterbrechungspunkten und separaten Korrekturverfahren (Merging). Dieser Abschnitt soll den in 7.2.3 beschriebenen Aufbau verdeutlichen und eine Anleitung zum erweitern des Funktionsumfangs geben.

7.4.1 Klasse AttributReplacer

Diese Klasse stellt die Implementation des Ersetzer-Werkzeuges aus Abschnitt 4.4.2.2 dar.

Es wird die *Klasse* *AttributReplacer* im *Paket* *Tools* erstellt, die das *Interface* *ITool* implementiert.

Die Funktionen zur Rückgabe der unterstützten Domäntypen können leer bleiben, da dieses Werkzeug wertunabhängig arbeitet. Jedoch muss der Name des Werkzeuges in der *getToolString()* Funktion aufgeführt sein. Namen von Werkzeugen sind eindeutig.

Die Funktion *getConfigShortText()* erzeugt aus einem String-Array, das die Werkzeugeinstellungen repräsentiert, einen kurzen Beschreibungstext, der in den Listenelementen angezeigt werden soll, wenn das Werkzeug konfiguriert wurde. Die Zuordnung der Schlüssel zu den Werten entspricht dabei der, vom Werkzeug bei der Werkzeugeinstellung selbst erstellten Zuordnungsvorschrift.

Diese Werkzeugeinstellungen werden mit der Oberfläche erstellt, die durch *editConfig()* beschrieben wird. Es ist darauf zu achten, dass das Werkzeug selber auf Eingaben dieser Maske reagieren muss. Die bei Eingaben werden geordnet als String-Array dem passenden Eintrag in der Prozessliste aus der *Klasse* *FrameworkCFG* zugeordnet. Hier würde die Einstellung der Tabelle, des Attributes, des Suchmusters, die Art des Suchmusters, die Ersetzung und ob eine Interaktion vorkommen soll, von der Oberfläche verlangt.

Die *runTool()* Methode führt schließlich das Werkzeug aus. Hier würde anhand der Einstellung der Interaktion entschieden, ob direkt ein SQL Befehl zur Datenmanipulation abgesetzt wird oder ob erst das Suchmuster in einer SELECT Anweisung auftritt und eine Interaktion anfordert. Eine Interaktion veranlasst die Methode im aktuellen Zustand

anzuhalten und an die übergeordnete Klasse mit Hilfe der *parentAction()* Methode einen Interaktionsdialog einzufordern. Dieser Dialog wird mit Hilfe der *interactionPanel()* Methode für das Werkzeug erzeugt.

Intern verwenden wir immer die REGEXP Funktionalität von Oracle, mit der wir SQL Anfragen mit regulären Ausdrücken schreiben können. Die Wahl nach der Art des Suchmusters bestimmt lediglich, ob das Suchmuster bereits ein regulärer Ausdruck ist. Für die SELECT Anfrage nutzen wir REGEXP_LIKE. Die Datenmanipulation kann in einem Ausdruck geschehen:

```
UPDATE SET Attribut = REGEXP_REPLACE(Attribut,Suchmuster,Ersetzung)
WHERE REGEXP_LIKE(Attribut,Suchmuster)
```

Die folgende Abbildung veranschaulicht diesen Ablauf:

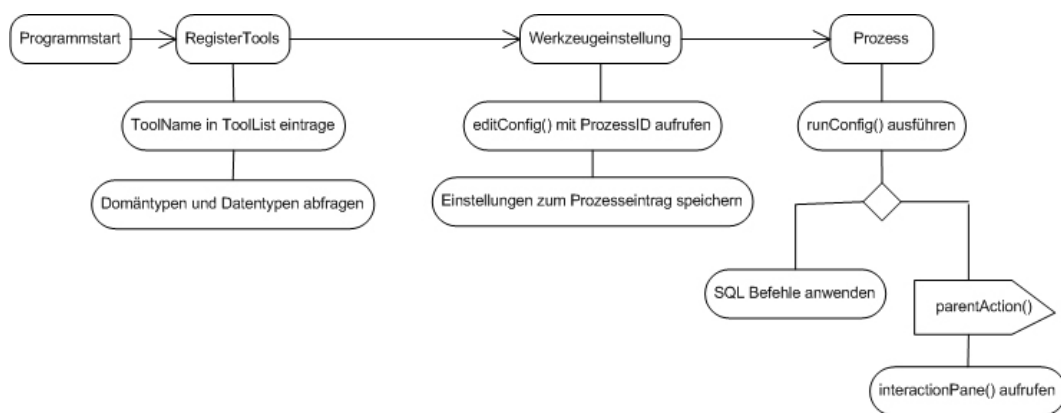


Abbildung 7.7: Ablauf AttributReplacer

Um das neu erstellte Werkzeug im Programm verwenden zu können, wird die Zeile „*Controller.addTool(new AttributReplacer());*“ in die Klasse *RegisterTools* eingetragen.

7.4.2 Klasse TupelDuplicate

Diese Klasse stellt die Implementation des Tupel-Duplikatbereinigers aus Abschnitt 4.4.1.1 dar.

Der Name und der „ConfigShortText“ werden gebildet wie schon unter 7.4.1. Für *getSimNames()* erstellen wir hier ein String-Array, in dessen Feldern wir die verfügbaren Domäntypen eintragen, die dieses Werkzeug zusätzlich zu den Standarddatentypen verarbeiten kann. Ebenso erstellen wir ein SimTypes-Array für *getSimTypes()*. In dessen Feldern tragen wir, in gleicher Reihenfolge, alle, den Domäntypen entsprechenden Datentypen, ein. Hier geben wir den Domäntyp „ISBN“ an und weisen diesem den Datentyp „NUMBER“ als Äquivalent zu.

Die *editConfig()* Methode erzeugt wieder die Oberfläche für die Werkzeugeinstellung. Hier weisen wir dem Werkzeug die aus 4.4.1.1 verlangten Parameter zu und behandeln referenzierte Tabellen.

Mit der *runTool()* Methode werden nun die benötigten Prozeduren, Funktionen und temporäre Tabellen auf die Datenbank übertragen. Anschließend wird für jede Tabelle

der Werkzeugeinstellung deren Liste der Attributeigenschaften generiert und die Prozeduren angestoßen, bis alle Tabellen abgearbeitet wurden. Dabei werden eventuell verwaiste Schlüssel korrigiert werden müssen. Dazu wird in den Werkzeugeinstellungen festgehalten, ob es zu der Tabelle Verweise gibt.

Tritt währenddessen das Unterbrechungssignal auf, so wird nach der Ausführung der letzten aktiven Prozedur das Werkzeug verlassen. Dazu wird vor dem Start der nächsten Prozedur geprüft, ob das Unterbrechungssignal gesetzt wurde. Die temporären Daten bleiben erhalten da gemäß 4.4.1.1 ein stabiler Zustand erreicht wurde.

Eine Interaktion wird wie unter 7.4.1 ausgelöst. Was für eine Interaktion angezeigt wird, bestimmt das Werkzeug durch interne Variablen.

Das Merging wird hier mit den Besonderheiten der Tupel-Ebene verwendet. Da die Merging-Prozedur vom Werkzeug erzeugt wird, hat dies keine Auswirkung auf andere Werkzeuge.

Die Wahl des Ähnlichkeitsmaßes geschieht dadurch, dass die Funktion zur Erzeugung des Ähnlichkeitsfaktors aus den Attributeigenschaften das passende Maß ermittelt. Es wurden also alle erzeugenden Funktionen übertragen, und die Wahl erfolgt während der Laufzeit.

7.5 Implementation von Verfahren

Bei den hier vorgestellten Verfahren handelt es sich um die Verfahren, deren Implementierung die Unterschiede in der Realisierung der zuvor vorgestellten Verfahren verdeutlicht. Weitere, zuvor vorgestellte Verfahren, lassen sich analog oder trivial implementieren.

7.5.1 Wertunabhängige Verfahren

Diese Verfahren können wir in der Regel durch eine geschickt generierte SQL Anfrage sehr einfach realisieren. Dazu wird die Bedingung, nach der eine Aktion ausgeführt werden soll, als Teil der Filterkette der SQL übersetzt. Die Aktion selber definiert den SQL Befehl und stellt den Funktionsteil der SQL dar.

Zum Beispiel kann das **Ersetzen** von *Wert A* durch *Wert B* geschrieben werden als:

UPDATE SET *Wert=B* WHERE *Wert=A*

Beispiel 7.1: Ersetzung von Werten durch wertunabhängiges Verfahren

Die Anwendung einer solchen Implementierung haben wir bereits im Abschnitt 7.4.1 gesehen.

7.5.2 PC-Filter

PC-Filter wird nach Abschnitt 4.4.1.1 in mehrere Blöcke aufgeteilt. Hinter jedem Block müssen temporäre Dateien gespeichert werden. Das lässt sich erreichen, indem wir das Verfahren als einzelne Prozeduren, entsprechend der Blockaufteilung in Abbildung 3.1,

implementieren. Zusätzlich implementieren wir PC-Filter als PL/SQL Prozeduren. Dadurch wird die Schnittstelle zur Datenbank nicht unnötig belastet. Um den Datenbestand nicht unnötig zu vergrößern, verwenden wir das von Oracle zur Verfügung gestellte RowID Attribut. Dieses beinhaltet eine eindeutige Identifikationsnummer einer Zeile einer Tabelle. Diese werden von Oracle automatisch indexiert. Der Ablauf entspricht der in Abschnitt 3.5 beschriebenen Vorgehensweise.

Es werden folgende temporäre Tabellen benötigt:

- tmpPartition: Tabelle für Partitionen, Anchor-Records (AR) und Ähnlichkeiten zum AR
- tmpPCR: Tabelle zum Verwalten der Delimiting-Records (DR)
- tmpPCRList: Tabelle zum Generieren der PartList
- tmpPCG: Tabelle für den PCG
- tmpDuplicate: Ergebnistabelle mit Duplikatpaaren
- tmpAttribute: Zuweisung von Daten-/Domaintypen zu Attributen

Die tmpAttribute Tabelle wird aus den Werkzeugeinstellungen generiert und wird für die Zuweisung eines Ähnlichkeitsmaßes gebraucht.

Die erste Prozedur sortiert die Tupel gemäß der Sortiervorgabe und ordnet allen RowIDs eine Partition zu. Zeitgleich werden die AR und DR Tupel bestimmt. Diese Zuordnungen werden in der tmpPartition und tmpPCR Tabelle abgelegt. Mit der Bestimmung der DR Tupel kann auch die PartList zu dem Tupel in tmpPCRList abgelegt werden.

Die zweite Prozedur berechnet die Ähnlichkeitsfaktoren zum AR und legt diese mit in tmpPartition ab.

Die dritte Prozedur wendet nun die Eigenschaft 4 von PC-Filter auf die in tmpPCRList abgelegten PartListen an. Das Ergebnis wird wieder in tmpPCRList gespeichert.

Die vierte Prozedur erstellt den PCG aus den PartListen in tmpPCG.

Die fünfte Prozedur sucht innerhalb der Partitionen nach Duplikaten und legt diese in tmpDuplicate ab.

Die sechste Prozedur durchläuft die Knoten des PCG aus tmpPCG und sucht nach Duplikaten und legt diese in tmpDuplicate ab.

Die Ähnlichkeitsmaße werden als PL/SQL Funktionen implementiert. Dadurch können interne Speicherstrukturen der Datenbank genutzt werden. Das für ein Attribut nötige Ähnlichkeitsmaß wird mit einer Funktion bestimmt, die die tmpAttribute Tabelle ausliest und die passende Funktion ausführt. PC-Filter verweist auf diese Auswahlfunktion und ist damit von der Konfiguration der Tabelle weitgehend unabhängig.

Jede der sechs Prozeduren erzeugt einen stabilen Endzustand, nach der die Abarbeitung pausiert werden kann. Die Prozeduren werden von der Werkzeugklasse angestoßen.

7.5.3 Mergingverfahren

Ziel des Verfahrens ist es, aus den gefundenen Duplikatpaaren durch geschicktes Abfragen der Duplikattabelle, eine Menge von Cluster von duplizierten Tupeln zu durchlaufen. Die Duplikatpaare liegen als RowID Paare vor. Das erste Ziel ist es, anhand einer RowID alle direkt oder indirekt verbundenen RowID zu finden. Dazu fassen wir die Tabelle als ungeordneten Graph auf und suchen mit der Tiefensuche zu einem Knoten alle verbundenen Knoten.

```

Für jedes (RowID,PartnerID) Paar aus Duplikattabelle
  Wenn RowID und PartnerID noch nicht in Ergebnistabelle
    Füge Paar in Ergebnistabelle ein
  Sonst: PartnerID in Ergebnistabelle
    Wähle ein Paar aus der Ergebnistabelle mit PartnerID
    Wenn PartnerID erstes Element vom Paar
      Füge das Paar (PartnerID,RowID) in Ergebnistabelle ein
    Sonst: PartnerID zweites Element vom Paar
      Füge das Paar (Erstes Element, RowID) in Ergebnistabelle ein
  Sonst: RowID in Ergebnistabelle
    Wähle ein Paar aus der Ergebnistabelle mit RowID
    Wenn RowID erstes Element vom Paar
      Füge das Paar (RowID,PartnerID) in Ergebnistabelle ein
    Sonst: RowID zweites Element vom Paar
      Füge das Paar (Erstes Element, PartnerID) in Ergebnistabelle ein

```

Auflistung 7.1: Merging: Clusterbildung

Dieser Teil des Mergings lässt sich aus einer Reihe von Abfragen als PL/SQL schreiben. Für die Umsetzung nach Auflistung 7.1 muss gelten, dass keine doppelten Einfügungen erlaubt sind und dass die Elemente der eingefügten Paare nicht gleich sind.

Anschließend wird auf Anwendungsebene ein Cluster gewählt und für jedes Attribut das Referenzwertverfahren gestartet. Dabei wird mitgeführt, in welchem Cluster und in welchem Attribut sich das Verfahren gerade befindet. Die Rückgaben des Referenzwertverfahrens werden in der Ersetzungstabelle **tmpMerging** abgelegt. Dabei wird jedem Attribut der Referenzwert zugeordnet. Diese Zuordnung kann dann in eine automatisch generierte UPDATE Anweisung umgesetzt werden.

```
UPDATE SET Attribut = ZugeordneterWert WHERE RowID=aktuelle-ID
```

Beispiel 7.2: Auswertung der Ersetzungstabelle

Wenn das Merging als Teil einer Duplikatbereinigung auf Tupel-Ebene gestartet wurde, also explizit durch das Werkzeug „Tupel-Duplikatbereiniger“, kann das Ersetzen und Entfernen duplizierter Tupel vereinfacht werden, indem für das Originaltupel die *den aktuellen Cluster beschreibende RowID* verwendet wird. Es werden nur in diesem Tupel

Attributwerte ersetzt und anschließend alle dem Cluster zugeordneten Tupel außer dem Original gelöscht.

Auf Attribut-Ebene müssen alle mit den RowIDs adressierten Tupel angepasst werden und das Löschen entfällt. Da diese letzten Operationen auf Anwendungsebene geschehen, wird das Verfahren je nach Werkzeug entsprechend anders implementiert.

7.5.4 Referenzwertverfahren

Das Referenzwertverfahren ist als PL/SQL Funktion implementiert. Es wird eine SELECT Anfrage übergeben, die in einem Cursor geöffnet wird. Dadurch ist die Funktion unabhängig vom Zustand temporärer Tabellen. Es wird davon ausgegangen, dass immer nur ein Attribut ausgewertet werden soll. Entsprechend wird die Attributeigenschaft direkt übergeben. Das Verfahren kann entweder Gebrauch von der tmpAttribute Tabelle machen oder vom Werkzeug mit einem festen Ähnlichkeitsmaß erstellt werden.

Die im Cursor geöffnete Anfrage wird auf die, nach Abschnitt 4.6 beschriebenen Arten, untersucht. Inwieweit alle Arten unterstützt werden, hängt vom gewählten Werkzeug ab. Das Grundprinzip für eine automatische Erkennung lautet dabei:

```

Für alle Attributwerte der Query
  zähle Auftreten des Attributwertes
Suche Attributwerte mit der höchsten Auftrittszahl
Wenn Anzahl der gefundenen Attributwerte > 1
  Fehler: Benutzerinteraktion benötigt
Teile höchste Auftrittszahl durch Gesamtzahl an Attributwerten
Wenn die Division eine vorgegebene Schranke überschreitet
  gebe Attributwert mit höchster Auftrittszahl zurück
sonst
  Fehler: Benutzerinteraktion benötigt

```

Auflistung 7.2: Referenzwert: automatische Bildung

Der gefundene Referenzwert wird zurückgegeben. In den, aus Kapitel 4 vorgestellten Werkzeugen, wird das Referenzwertverfahren immer über Merging gestartet. Das Merging startet das Referenzwertverfahren von der Anwendungsebene. Die Verfahren agieren also unabhängig voneinander. Wird eine Interaktion benötigt, so wird diese durch das Merging behandelt. Die Kontrolle über die Cluster bleibt auf der Anwendungsseite. Das Referenzwertverfahren kann folglich mit beliebigen Clusterzusammensetzungen wiederholt aufgerufen werden, ohne dass sich der Prozesszustand ändert. Dies ermöglicht es eine Clusterkorrektur anzubieten. Da sich durch das Referenzwertverfahren der Zustand nicht geändert hat, kann dieses erneut aufgerufen werden.

7.5.5 Modifizierte Field-Similarity

Das modifizierte Field-Similarity Verfahren ist mit vier PL/SQL Funktionen implementiert. Dabei setzen diese sich zusammen aus dem Verfahren zum Booleschen-Vergleich,

das Field-Similarity Verfahren, dem Q-Grams $q=2$ Verfahren und der Funktion, die den Ablauf gemäß Abschnitt 4.7.4 realisiert.

Zunächst müssen die Wortmengen erstellt werden. Dies lässt sich mit den *INSTR* und *SUBSTR* Funktionen lösen. Wir erhalten zwei PL/SQL Tabellen, die die Wortmengen darstellen. Ebenso können wir die Anzahl der Wörter zählen.

```

Gesamtzahl:=0
i := 0
fertig := falsch
Trennzeichen := ' '
Solange nicht fertig
  i:=i+1
  Finde das i'ten Auftreten des Trennzeichens in Eingabe A
  Ist kein i'tes Trennzeichen vorhanden dann
    fertig := wahr
  Wenn nicht fertig
    Bestimme nächstes Auftreten des Trennzeichens oder Satzende
    Gesamtzahl := Gesamtzahl+1
    TabelleA(i) := Teilfolge zwischen den beiden Auftrittspunkten.
i := 0
fertig := falsch
Solange nicht fertig
  i:=i+1
  Finde das i'ten Auftreten des Trennzeichens in Eingabe B
  Ist kein i'tes Trennzeichen vorhanden dann
    fertig := wahr
  Wenn nicht fertig
    Bestimme nächstes Auftreten des Trennzeichens oder Satzende
    Gesamtzahl := Gesamtzahl+1
    TabelleB(i) := Teilfolge zwischen den beiden Auftrittspunkten.

```

Auflistung 7.3: Implementierung Atomic-Strings

Das Durchlaufen der Wortmengen übernimmt die steuernde Funktion, dadurch können die vergleichenden Funktionen sich auf den Vergleich von zwei einfachen Zeichenketten beschränken.

Der Boolesche-Vergleich überprüft die Zeichenketten lediglich auf exakte Übereinstimmung. Je nach Werkzeugimplementierung kann diese Funktion auch Teil der steuernden Funktion sein.

Das Field-Similarity Verfahren verwendet PL/SQL Tabellen, um die Zeichen zu zählen. Dabei wird von der Möglichkeit der Oracle 10 Gebrauch gemacht, dass PL/SQL Tabellen auch über Zeichenketten indiziert werden können. Dazu sei gesagt, dass sich eine PL/SQL Tabelle wie ein assoziativer Vektor verhält.

Zunächst wird die erste Zeichenkette durchlaufen und zu jedem Zeichen ein Vektorindex erstellt. Der Wert des Vektors am Vektorindex ist die Auftrittsnummer. Wird ein Vektorindex neu angelegt, sei die ursprüngliche Auftrittsnummer 0. Bei bereits existierendem Vektorindex wird die Nummer erhöht.

Anschließend wird die zweite Zeichenkette durchlaufen und für jeden Vektorindex der Wert um einen reduziert. Solange der Wert des Vektors am Vektorindex nicht unter 0 fällt, wird ein Zähler um einen erhöht. Dieser Zähler stellt den Betrag der Schnittmenge dar. Dies wird dadurch klar, dass zu jedem Vektorindex, dessen Auftrittsnummer reduziert wird, ein passendes Zeichen in der zweiten Zeichenkette gefunden wurde. Gerät die Auftrittsnummer hingegen unter 0, so ist das Zeichen in der zweiten Zeichenkette zusätzlich enthalten und wird nicht mit zum Zähler gerechnet. Ähnlich, wenn nicht alle Auftrittsnummern 0 erreichen, hat der erste Zeichenkette mehr Zeichen. Im Folgenden sei der Vektor als „*Auftrittsnummer*“ bezeichnet. Der Zähler zum Ermitteln des Betrags der Schnittmenge sei „*Schnitt*“.

```

Schnitt := 0
Für alle Zeichen c aus Eingabe A
    Erhöhe Auftrittsnummer[c] um 1
Für alle Zeichen c aus Eingabe B
    Verringere Auftrittsnummer[c] um 1
    Wenn Auftrittsnummer[c] >= 0
        Schnitt := Schnitt + 1
Rückgabe := Schnitt / Maximum(Länge A, Länge B)

```

Auflistung 7.4: Implementierung von Field-Similarity

Q-Grams mit $q=2$ lässt sich ähnlich wie Field-Similarity implementieren. Der Unterschied ist, dass je zwei Zeichen einen Vektorindex definieren. Auf Start- und Endzeichen können wir verzichten, wenn wir das erste und das letzte Zeichen separat in den Vektor eintragen.

```

Schnitt := 0
Für erstes Zeichen c aus A
    Setze Auftrittsnummer[c] auf 1
Für letztes Zeichen c aus A
    Setze Auftrittsnummer[c] auf 1
Für alle 2 Zeichen ab aus Eingabe A
    Erhöhe Auftrittsnummer[ab] um 1
Für alle 2 Zeichen ab aus Eingabe B
    Verringere Auftrittsnummer[ab] um 1
    Wenn Auftrittsnummer[ab] >= 0
        Schnitt := Schnitt + 1
Für erstes Zeichen c aus B
    Verringere Auftrittsnummer[c] um 1
    Wenn Auftrittsnummer[c] >= 0
        Schnitt := Schnitt + 1
Für letztes Zeichen c aus B
    Verringere Auftrittsnummer[c] um 1
    Wenn Auftrittsnummer[c] >= 0
        Schnitt := Schnitt + 1
LenAB := Anzahl der erzeugten Indexe
Rückgabe := Schnitt / (LenAB)

```

Auflistung 7.5: Implementierung von Q-Grams $q=2$

Kapitel 8

Ausblick

8.1 Erweiterung von Tupeln auf Objekte

Hier soll das Objekt als neuer Datengegenstand näher erläutert werden.

Bei der normalen Duplikaterkennung folgen wir gemäß 2.2 rekursiv den Verweisen und korrigieren die entstandenen verwaisten Schlüssel nach erfolgter Duplikatbereinigung in den referenzierten Tabellen. Der Effekt ist, dass fremde Tupel, die auf stark ähnliche Tupel mit Zusatzinformationen verweisen, nach Ausführung der Bereinigung nur noch auf die gleichen Zusatzinformation zeigen. Nämlich die, die durch das Merging aus den stark ähnlichen Zusatzinformationen hervorgegangen sind. Im schlechtesten Falle kann das Merging sogar zu Fehlinformationen im Verbund mit dem verweisenden Datensatz führen. Als Beispiel nehmen wir die Tabellen „Lebensmittel“ und „Typ“ an. Verweise nun der Eintrag „Brot“ und „Mehl“ auf zwei Einträge „Roggen“, so würde nach dem Ansatz des Folgens der Verweise nur noch ein Eintrag „Roggen“ übrig bleiben, auf den beide Lebensmittel verweisen. Ändert sich nun der Mehltyp auf „Weizen“, darf der nun referenzierte Eintrag „Roggen“ nicht einfach geändert werden.

Beim Objekt soll es sich hier nun um eine Kombination aus Tupel und Beziehung handeln. Das Verbindungselement stellt dabei die Beziehung zwischen den Tupeln dar. Der referenzierte Tupel wird als „Beziehungsattribut“ an den verweisenden Tupel angehängt. Die Beziehung zwischen den Tupeln wird damit zum künstlichen Datentyp und der referenzierte Tupel der Attributwert. Diesem neuem Attribut kann ein Ähnlichkeitsmaß zugeordnet werden. Dadurch wird sowohl der verweisende als auch der referenzierte Tupel als Einheit bewertet. Wird nun ein Objekt als Duplikat erkannt und behoben, bleibt die ursprüngliche Datenredundanz zu referenzierten Tupeln erhalten.

8.2 Erweiterung der Werkzeuge

Die hier dargestellten Werkzeuge entsprechen nur einem kleinen Teil des Möglichen. So kann als Attributwertvergleich zur wertabhängigen Feglerkorrektur auch das Verfahren von [MB06] eingesetzt werden. Zudem können weitere Attributwerkzeuge für neue Domaintypen erstellt werden. So haben wir bisher nur relativ generische Fälle behandelt, nicht aber die Schwierigkeiten beim Vergleich von räumlichen Daten berücksichtigt. Ein

solcher Vorgang benötigt Ähnlichkeitsmaße, die die Geometrie der Daten auswerten können. Darunter könnte man auch Vergleichsverfahren für Realms einordnen.

Die Anwendung beschäftigt sich mit dem Bereinigen einer Tabelle und den von ihr referenzierten Tabellen. Nicht realisiert wurde das angedachte Bereinigen von Tabellen mit Join-Bedingungen, die nicht zwangsläufig einen Foreign-Key besitzen. Dazu müsste der Anwendungspart, der sich mit der Auflösung der Foreign-Keys beschäftigt, darauf erweitert werden, die Join-Bedingung zu analysieren und die entsprechende Bereinigungsreihenfolge herzustellen.

8.3 Sicherung der Datenqualität

Mit CHECK Constraints können Attributen Vorgaben zum Wertebereich gemacht werden. Dies ist eine Eigenschaft der Tabelle und wirkt immer. Für mehr Flexibilität können Trigger verwendet werden. Dies sind PL/SQL Strukturen, die auf datenmodifizierende Anfragen reagieren können. Mit einem Trigger ist es ebenfalls möglich, die in dieser Arbeit entwickelten PL/SQL Funktionen zur Bestimmung von Ähnlichkeitsfaktoren zu nutzen. Damit ist es zum Beispiel möglich, bei Einfügeoperationen von einem Namen zu prüfen, ob ein Ähnlicher bereits in der Tabelle vorhanden ist. Dies ist nützlich, um Daten abzuhalten, die nicht schon in der Tabelle sind oder gänzlich neu sind. So werden Tippfehler schon per Trigger geblockt.

Da diese Methode sehr radikal ist können auch externe Skripte diese Funktionen nutzen. Damit ist es möglich für eine Eingabemaske, zu den bereits eingegebenen Daten, ähnliche Werte aus der Datenbank anzeigen zu lassen. Damit kann das Raten nach der richtigen Schreibweise vermieden werden.

Anhang A

API

A.1 Paket Configuration

Klasse Controller

public static void registerTools()

Führt die *RegisterTools.register()* Methode aus und erstellt die Liste der Domäntypen.

public static void clearConfig()

Setzt die gesamte Konfiguration zurück.

public static void clearToolConfig()

Setzt die Liste der registrierten Werkzeuge zurück.

public static ITool getTool(String id)

Gibt das Werkzeug mit der Werkzeug-Id *id* zurück.

public static void addTool(ITool tool)

Fügt ein Werkzeug der Konfiguration hinzu.

public static Object[] getToolArray()

Gibt die Liste der Werkzeuge als Objekt-Array zurück.

public static Object[] getSimpleQueryArray(String query)

Führt eine SQL Anweisung aus, deren erster Rückgabewert als Objekt-Array zurückgegeben wird.

public static String getDbUser()

Gibt den Benutzernamen des derzeit angemeldeten Benutzers zurück.

public static void connectDB(String user, String pass)

Stellt eine Verbindung mit der Datenbank her.

public static void closeDB()

Schließt die Datenverbindung.

public static boolean isConnected()

Gibt den Verbindungszustand zurück.

public static FrameworkCFG getFWConfig()

Gibt die Instanz der Klasse *FrameworkCFG* zurück.

public static void fillAttrList(ITableList tlsTableList)

Sucht zu einer Tabelle deren Attribute und erstellt daraus eine Liste.

public static void saveConfig(String saveName)

public static void saveState(String saveName)

public static void loadConfig(String loadName)

public static void loadState(String loadName)

Speichert, bzw. lädt die Konfiguration oder Prozesszustand. Siehe hierzu das *Interface IXMLWrapper*.

Klasse FrameworkCFG

Die Klasse besteht aus einer Vielzahl an Getter- und Setter-Methoden. Hier sei eine Auswahl an nicht trivialen Methoden gegeben.

public void addProcess(String toolID)

Fügt einen neuen Prozess zur Prozessliste mit Verweis auf die Werkzeug-Id *toolID*.

public void removeProcess(int key)

Entfernt den Prozess mit der Id *key*.

public void setToolOptions(int przKey, String[] options)

Setzt die Werkzeugeinstellungen für eine Prozess-Id.

public String[] getToolOptions(int przKey)

Findet zu einer Prozess-Id die zugehörigen Werkzeugeinstellungen.

public String getToolID(int key)

Findet zu einer Prozess-Id die zugehörige Werkzeug-Id.

public Object[] getProcessArray()

Gibt die Prozessliste als Object-Array zurück.

public void clearCurState()

Setzt den Prozesszustand zurück.

Klasse IProcessState

public boolean isResumeable()

Gibt zurück, dass der Prozess fortsetzbar ist.

public void setResumeable(boolean resume)

Setzt das Flag, nach dem die Fortsetzbarkeit entschieden wird.

public int getProzessID()

Gibt die aktuellen Prozess-Id zurück.

public void setProzessID(int prozessID)

Setzt die aktuelle Prozess-Id.

public int getToolState()

Gibt den Prozesszustand des Werkzeugs als Nummer zurück.

public void setToolState(int toolState)

Setzt den Prozesszustand als Nummer.

public String[] getToolOptions()

Gibt vom Werkzeug gesetzte Prozessdaten zurück.

public void setToolOptions(String[] toolOptions)

Setzt die Prozessdaten.

public void addToolOption(int key, String toolOption)

Setzt gesondert adressierte Prozessdaten auf den Wert von *toolOption*.

public String getProcessLog()

Gibt das Prozesslog zurück.

public void setProcessLog(String processLog)

Setzt das Prozesslog.

Interface IXMLWrapper

public void writeConfig(String saveName, FrameworkCFG fCfg)

Speichert die in *fCfg* abgelegte Konfiguration mit dem Dateinamen *saveName*.

public void readConfig(String loadName, FrameworkCFG fCfg)

Lädt die Konfiguration mit dem Dateinamen *loadName* nach *fCfg*.

public void writeState(String saveName, FrameworkCFG fCfg)

Speichert den in *fCfg* abgelegten Prozesszustand mit dem Dateinamen *saveName*.

public void readState(String loadName, FrameworkCFG fCfg)

Lädt den Prozesszustand mit dem Dateinamen *loadName* nach *fCfg*.

A.2 Paket View

Abstrakte Klasse StandardView

public abstract Window getFrame()

Gibt das Fensterobjekt zurück, dass mit der Klasse instanziiert wurde.

public JButton createJButton(String title, ActionListener aListener, String command)

Erzeugt einen JButton mit dem Titel *title* und fügt gleichzeitig den ActionListener und den Commandstring hinzu.

public JMenuItem createJMenuItem(String title, ActionListener aListener, String command)

Erzeugt ein JMenuItem mit dem Titel *title* und fügt gleichzeitig den ActionListener und den Commandstring hinzu.

public abstract void parentAction(String command, Object[] args)

Wird von übergeordnetem Objekt aufgerufen, wenn eine Rückmeldung erforderlich ist.

Abstrakte Klasse InitView

public abstract void unload();

Vernichtet das Fenster.

public abstract void pack();

Wendet das Layout auf die Komponenten an.

public abstract void show();

Zeigt das Fenster.

public abstract void hide();

Versteckt das Fenster.

public abstract void centerFrame();

Erzeugt ein Fenster variabler Größe in der Bildschirmmitte.

public abstract void setFrame();

Erzeugt ein Fenster mit Standardabmessungen.

public abstract void setFrame(int x, int y);

Erzeugt ein Fenster mit den Abmessungen x und y .

public abstract void setFrameScreen();

Erzeugt ein Fenster mit der halben Bildschirmauflösung als Abmessung.

public abstract void disableFrame();

public abstract void enableFrame();

Sperrt , bzw. aktiviert das Fenster.

public abstract void disableResize();

Verhindert das Ändern der Fenstergröße mit der Maus.

public abstract void enableResize();

Erlaubt das Ändern der Fenstergröße mit der Maus.

Klasse GBCHelper

public void resetGBC()

Setzt das gesamte Layout zurück.

public void addToPane(Component com)

Fügt eine Komponente dem Layout hinzu.

```
public void noFill()  
public void fillHor()  
public void fillVer()  
public void fillBoth()
```

Legt fest, in welche Richtung Komponenten ihre Fläche ausfüllen.

```
public void setGridX(int x)  
public void setGridY(int y)
```

Legt die Koordinaten einer Komponente im Layout fest.

```
public void resetGridsize()  
public void setGridwidth(int width)  
public void setGridheight(int height)
```

Legt die Ausdehnung einer Fläche fest.

```
public void resetWeigth()  
public void setWeightX(double x)  
public void setWeightY(double y)
```

Legt die Dehnungsverteilung fest.

```
public void setIPadX(int x)  
public void setIPadY(int y)  
public void resetPad()
```

Legt fest, wie viel zusätzlichen Platz eine Komponente braucht (innerer Rand).

```
public void setInsets(int top, int left, int bottom, int right)  
public void resetMargin()  
public void setMarginTop(int px)  
public void setMarginLeft(int px)  
public void setMarginRight(int px)  
public void setMarginBottom(int px)
```

Diese Methoden legen die Abstände zwischen Komponenten fest.

```
public void anchorCenter()  
public void anchorLeft()  
public void anchorRight()  
public void anchorTop()  
public void anchorBottom()  
public void anchorTopLeft()  
public void anchorTopRight()
```

Legt die Ausrichtung einer Komponente in einer Fläche fest.

A.3 Paket Tools

Beinhaltet die Werkzeugklassen

Interface ITools

public String getToolString()

Gibt den Namen des Werkzeugs zurück. Namen müssen eindeutig sein.

public String[] getSimNames()

Gibt die Namen der unterstützten Domäntypen zurück. Das Array muss die gleichen Indizes wie die Rückgabe von *getSimTypes()* haben.

public SimTypes[] getSimTypes()

Gibt die zugrunde liegenden Datentypen der Domäntypen zurück. Das Array muss die gleichen Indizes wie die Rückgabe von *getSimNames()* haben.

public String getConfigShortText(String[] options)

Erstellt aus einem String-Array von Werkzeugeinstellungen einen Beschreibungstext für die Listendarstellung.

public void editConfig(int processID, JPanel configPanel)

Fügt die Oberfläche der Werkzeugeinstellungen an das *configPanel* an. Der Parameter *processID* gibt an, zu welchem Prozess der Prozessliste die Einstellungen abgelegt werden sollen.

public void interactionPane(int processID, JPanel interPanel)

Fügt die Oberfläche der Interaktion an das *configPanel* an. Der Parameter *processID* gibt an, in welchem Prozess sich die Anwendung gerade befindet. Mit dieser Information kann die Interaktionsoberfläche von dem Prozesszustand Gebrauch machen.

public void runTool(int processID, StandardView parent)

Führt ein Werkzeug aus. Der Parameter *processID* gibt an, in welchem Prozess sich die Anwendung gerade befindet. Damit kann vom Prozesszustand Gebrauch gemacht werden. Der Parameter *parent* zeigt auf die Klasse, deren *parentAction()* Methode auf Interaktionsanfragen reagiert.

Klasse RegisterTools

public static void register()

Wird von der Klasse *Controller* aus dem Paket *Configuration* aufgerufen. Die Methode ist dazu gedacht, dass Werkzeuge an der Konfiguration angemeldet werden. Siehe auch *Controller.addTool(ITool tool)*.

Abbildungsverzeichnis

2.1	Nachführende Behebung von duplizierten Tupeln mit Verweisen	8
2.2	Entity-Relationship-Model	17
3.1	PC-Filter Übersicht	24
3.2	Ähnlichkeitsfaktoren am Dreieck	25
3.3	Lower und Upper Bound am Zahlenstrahl	26
3.4	PCG Beispiel	28
4.1	Architekturebenen	32
4.2	MVC Abhängigkeitsbeispiel	33
4.3	Funktionsblöcke von Werkzeugen	35
4.4	Anwendung Funktionsblöcke	37
4.5	PC-Filter Funktionsblöcke	41
4.6	Vergleichsverfahren Ablaufdiagramm	47
5.1	Hauptoberfläche: Konfiguration	56
5.2	Konfiguration laden	57
5.3	Hinweisfeld	57
5.4	Datenbanklogin	58
5.5	Datenbankzugang	58
5.6	Tabellenkonfiguration	59
5.7	Attributkonfiguration	60
5.8	Prozessliste	61
5.9	Werkzeugeinstellung: kein gewähltes Werkzeug	62
5.10	Werkzeugeinstellung: Tupel-Duplikatbereiniger	63
5.11	Werkzeugeinstellung: Referenzwerte	64
5.12	Werkzeugeinstellung: Verweise	65
5.13	Hauptoberfläche: Prozess	66
5.14	Interaktion: Referenzwerte	67
5.15	Interaktion: Clusterkorrektur	68

6.1	Schlagworttabelle wählen	69
6.2	Attribut-Duplikatbereiniger einstellen	70
6.3	Laufender Prozess	71
6.4	Zuviele ähnliche Werte	72
7.1	Paket-Diagramm	74
7.2	Klassendiagramm Hauptoberfläche	75
7.3	Klassendiagramm 1 von Paket View	76
7.4	Klassendiagramm 2 von Paket View	78
7.5	Klassendiagramm Paket Configuration	79
7.6	Klassendiagramm Paket Tools	80
7.7	Ablauf AttributReplacer	84

Beispielverzeichnis

2.1	Clusterkorrektur bei unzureichender Duplikaterkennung	16
2.2	Dupliziertes Schlagwort	18
3.1	Zeichendreher mit $q=2$	21
3.2	Austausch von zwei Zeichen mit $q=2$	21
3.3	unähnliche Zeichenfolge mit $q=2$	21
3.4	unähnliche Zeichenfolge mit $q=3$	21
3.5	unähnliche Zeichenfolge mit Field-Similarity	22
3.6	Beispiel für Dreiecksungleichung nach PC-Filter	26
4.1	Benutzerinteraktion im Ausführungsblock	38
4.2	Beispiel für Differenzbeträge	48
4.3	Modifizierte Field-Similarity für Sätze	50
4.4	Modifizierte Field-Similarity für Zeichendreher	51
4.5	Modifizierte Field-Similarity für Name-Vorname-Kombinationen	51
4.6	Modifizierte Field-Similarity bei gleichem Zeichenvorrat	51
7.1	Ersetzung von Werten durch wertunabhängiges Verfahren	85
7.2	Auswertung der Ersetzungstabelle	87

Listenverzeichnis

2.1	Bereinigung wertabhängiger Fehler der Attribut-Ebene	10
2.2	Bereinigung auf Tupel-Ebene	10
2.3	Einfügungen zur Bereinigung von Verweisen	11
3.1	Allgemeine Dreiecksungleichung	25
3.2	Dreiecksungleichung nach PC-Filter	25
4.1	Zusammenfassung der Modifizierte Field-Similarity	52
5.1	Aufruf auf Kommandozeile	56
7.1	Merging: Clusterbildung	87
7.2	Referenzwert: automatische Bildung	88
7.3	Implementierung Atomic-Strings	89
7.4	Implementierung von Field-Similarity	90
7.5	Implementierung von Q-Grams $q=2$	90

Tabellenverzeichnis

3.1	Vergleich der Ähnlichkeitsfaktoren	23
3.2	Vereinigung der PartListen pro Partition	28
3.3	PCR Beispiel	28

Literaturverzeichnis

- [Bra07] M. Bramer: *Principles of Data Mining*. Springer-Verlag, London, 2007.
- [BS06] C. Batini, M. Scannapieco: *Data Quality - Concepts, Methodologies and Techniques*, 1 edition. Data-Centric Systems and Applications, Springer-Verlag, Berlin, 2006.
- [EIV07] A. K. Elmagarmid, P. G. Ipeirotis, V. S. Verykios: Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 19:1 (2007), 1–16.
- [HM02] D. O. Holmes, M. C. McCabe: Improving Precision and Recall for Soundex Retrieval. In I. C. Society (ed.), *International Conference on Information Technology: Coding and Computing : April 8 - 10, 2002, Las Vegas, Nevada, USA*, IEEE Computer Soc., Los Alamitos, Calif. [u.a.], 2002, 22–27.
- [HS95] M. A. Hernández, S. J. Stolfo: The Merge/Purge Problem for Large Databases. In M. Carey, D. Schneider (eds.), *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, May 22-25, 1995*, SIGMOD Record 2, ACM Press, New York, 1995, 127–138.
- [HS98] M. A. Hernández, S. J. Stolfo: Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery* 2:1 (1998), 9–37.
- [Lev66] V. I. Levenshtein: Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10:8 (1966), 707–710.
- [MB06] A. Mazeika, M. H. Böhlen: Cleansing Databases of Misspelled Proper Nouns. In *Proceedings of the First Int. VLDB Workshop on Clean Databases, CleanDB 2006, September 11, 2006, Seoul, Korea*, 2006.
- [ME96] A. E. Monge, C. Elkan: The Field Matching Problem: Algorithms and Applications. In E. Simoudis, J. Han, U. M. Fayyad (eds.), *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, Calif., 1996, 267–270.
- [SLP02] S. Y. Sung, Z. Li, S. Peng: A fast filtering scheme for large database cleansing. In K. Kalpakis (ed.), *Proceedings of the Eleventh International Conference on Information and Knowledge Management, CIKM 2002 : November 4 - 9, 2002, McLean, Virginia, USA*, ACM Press, New York, NY, 2002, 76–83.

- [Ukk92] E. Ukkonen: Approximate String Matching with q-grams and Maximal Matches. *Theoretical Computer Science* 92:1 (1992), 191–211.
- [ZLBL04] J. Zhang, T. W. Ling, R. M. Bruckner, H. Liu: PC-Filter: A Robust Filtering Technique for Duplicate Record Detection in Large Databases. In F. Galindo, M. Takizawa, R. Traunmüller (eds.), *Database and Expert Systems Applications – 15th International Conference, DEXA 2004, Zaragoza, Spain, August 30 - September 3, 2004*, LNCS, Springer-Verlag, Berlin, 2004, 486–496.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit und die zugehörige Implementierung selbständig verfasst und dabei nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Hannover, 02. April 2009

Philipp Wolski